Specification of Formalized Software Patterns for the Development of User Interfaces

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

Abstract — **The aim of this paper is the development of specifications for a general analysis model for user interface patterns that can be applied in a model-based user interface development process. To accomplish this, we compile a detailed definition of what user interface patterns are and how they can be classified. Furthermore, we analyze how available methods and notations can be used for a pattern application in user interface development, based on two exemplary applications of the pattern "Advanced Search" in the formal notations UIML and UsiXML. From the resulting possibilities and limitations in identification, selection, instantiation and integration of user interface patterns, we derive specifications for a sufficient pattern description and development integration method: an exact definition, a metamodel, a specialized language, and, in practice, a repository or pattern management software.**

Keywords — user interface patterns; user interface development; pattern specifications; UIML; UsiXML.

I. INTRODUCTION

A. Motivation

The design and implementation of user interfaces is still a complex and resource-consuming task. In general, pattern-based software development is a means to more efficient implementation by applying reusable solutions for miscellaneous software design problem classes. In this regard, the use of software patterns in user interface development would offer generic solutions for recurring components of a user interface, depending on a certain interface paradigm. Navigation through tabs, for example, would be a feasible solution for the need to switch between complex sets of documents, websites, forms, etc. in graphical and touch user interfaces.

Currently, the application of such user interface patterns is situated only on an informal level with textual descriptions of common design solutions [1]. There is only limited research into generative, formalized user interface patterns, which can be applied for the automation of re-use of design solutions [1]. In this regard, methods for the development of user interfaces were introduced, starting at the stage of task or system models and matching user interface patterns with parts of these models [2][3].

However, we found no consistent suggestion of a pattern-based user interface design and implementation process, which combines a sufficient pattern repository, consisting of formalized user interface patterns, and an end-to-end solution of model-based pattern matching, selection, instantiation, and code generation. In addition, a generally accepted notation for user interface patterns is missing, which allows an abstract formulation of human-computer interaction components. Being transferable into concrete user interface-part descriptions and, finally, instantiable into source code, these abstract components could be deployed to form elements of real user interface patterns and thus facilitate reuse in GUI development.

B. Objectives

The aim of this paper is the development of a specification for a general analysis model that describes generative user interface patterns so that their common aspects can be identified and captured. This basic specification and its understanding are needed for the integration of methods that enable the matching and code generation based on the application of these patterns. We explicate how available methods and notations could be used for a user interface pattern repository or pattern manager. Moreover, we analyze the strengths and weaknesses of these existing assets and point out what better suitable methods and formats would have to be capable of. A sufficient solution for pattern-based user interface development should particularly meet the following criteria:

- reusability and variability of stored user interface patterns
- ability of user interface patterns to be composed in order to form a hierarchy of GUI components
- instantiation of user interface patterns into varying interface paradigms and types

Based on these criteria, we review the state of the art and describe a perspective on user interface patterns that paves the way for the specification of a sophisticated metamodel needed in model-based user interface development environments.

C. Structure of the Paper

In Section II, we analyze existing methods of user interface development and independent interface description languages. We also outline the current status of the application of user interface patterns in the development process. In Section III, we propose a definition and characterization of user interface patterns, their inclusion criteria and dimensions. We use this definition to establish and utilize a formalized pattern, advanced search, for the application of current methods and notations in Sections IV and V. We show the results and weaknesses of our work and derive requirements for a fully applicable formal pattern description language in Section VI. Finally, in Section VII

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-230-1

we conclude with specifications for formalized user interface patterns, which will meet the three criteria mentioned in our objectives.

II. Related Work

A. User Interface Development and Description

Today, the design and implementation of a software user interface mainly concentrates on the basic conditions and abilities of the before-chosen programming language and used software frameworks or libraries. After the general design of a user interface, the implementation is in focus, whether it is in Java Swing, HTML and CSS or C# and the XAML, to name only a few examples. While there has been a lot of research conducted on model-based user interface development, only a limited number of generic model concepts for a methodic interface design exist. One of these can be found in [4], where common steps of a user interface development process are explicated. Four model layers and corresponding transformations to derive user interface specifications from requirement models are proposed by Ludolph.

Another approach relies on a UML-based design of user interface software architecture [5]. Chlebek describes a comprehensive process and provides several perspectives onto the user interface development. Also, a special description language for the development process, which is independent from target source code, is used by him.

A greater number of platform-independent user interface description languages do exist. These languages are often XML-based and thus markup languages. Some of them have been developed for certain software projects or company-specific programming tools, such as XUL [6] and XAML [7]. Others, like UIML [8][9] and UsiXML [10], are results of research projects, but are rarely used in practice.

None of those generic concepts for interface development processes we found enabled the application of user interface patterns. Neither do independent user interface description languages have sufficient capabilities to store user interface patterns in their format. The GUI aspects described by these languages tend to be invariant and too concrete in specification [11] so that they do not provide any means to adapt the user interface to varying contexts. However, several special approaches for an integration of patterns into user interface development exist, which are outlined in the following subsection.

B. Pattern-based User Interface Development

Currently, there is no generally accepted definition of software patterns for user interface development. Instead, different concepts and terms exist, such as user interface patterns, user interface design patterns, or human-computer interaction patterns. Most of them refer to textual and graphical descriptions as solutions of a user interface design problem concerning mostly visual aspects and interaction concepts. These are termed descriptive user interface patterns [1]. Several libraries of descriptive patterns exist, such as [12][13][14]. Rarely do such descriptive pattern collections provide implementation details [15].

For a direct integration of reusable patterns for the user interface into a development environment, formal models or notations are needed, which enable a certain functionality and can be instantiated into certain model stages or source code, like design patterns. This variant is called generative pattern [1].

Generative patterns can be applied in a pattern-based user interface development process. One example of such a development process has been created by the University of Rostock in Germany [3][16][17][18][19][20]. Therein, model-based and pattern-driven design has been integrated by using several model layers (task, dialog, presentation and layout) to perform an identification and a selection, an instantiation and an integration of user interface patterns during the generation of the used models [3]. A tool has been developed, which supports this integrated development [3]. The user interface patterns are stored as fragments of the used models (“patterns in modeling”). They are used for more efficient modeling steps (“accelerating the design”) [18]. In a similar approach, an enhanced CASE tool was suggested, where user interface patterns are stored as class diagrams [2]. The static description of classes is then matched with the existing patterns, enabling a high level design of systems and their user interface. Identified classes can be replaced by the corresponding stored pattern, which is again a contribution to efficiency of the interface development process at the modeling level.

Other approaches go further to the generation of formal user interface description or source code [16][17]. Here, XUL is used to store formalized patterns, or a combination of PLML [21], UsiXML and additional components [22].

However, due to the used description languages, only one interaction paradigm is supported — the so-called WIMP (windows, icons, menus and pointer) interface typical for modern desktop computer and notebook operating systems. Furthermore, a major issue of the suggested integration of patterns in the development process is the need for manual retouching work. In this respect, the pattern instances have to be created manually by adapting them to their application context. In addition, not all kinds of patterns are supported. The occurrence of sub-patterns is the only relation between user interface patterns, which is dealt with in detail.

Starting with the definition itself, currently there is no consent to the arrangement of software patterns for the development of user interfaces, their structure and characteristics, as well as their relations among each other and to other software patterns.

III. User Interface Patterns: Definition and Characteristics

A generally acceptable definition of software patterns in user interface development should not only describe precisely what a pattern is and how it can be reused and adapted. Additionally, it should combine the several dimensions these patterns can be classified with. Thus, we propose the following definition:

In general, **user interface patterns** are software patterns, which can be applied for the specification, description and development of user interfaces.

As there is no common basis in literature for user interface pattern characteristics, the definition above is to be refined by our findings and arguments focused on the

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-230-1

compositional view on user interface patterns we gathered during our observations in an industry project. The argumentative perspective presented here leads to requirements for a formal definition of user interface patterns that can be implemented by a metamodel in future work. To establish a more detailed clarification we describe the aspects a user interface pattern basically consists of in the following sub-section.

A. *User Interface Pattern Aspects*

Firstly**,** a user interface pattern incorporates a stereotype but abstract **view**. This aspect defines the selection, arrangement and types of user interface controls. Regarding this aspect, the user interface pattern does not refer to certain GUI frameworks so that the view can be implemented using different languages and technologies. In addition, the view is abstract in order to allow its application in various contexts. The abstract manner of view is backed by other user interface pattern specification language sources. For instance, the "facets" and "Abstract Interaction Objects" of the "abstract UI model" in [1] imply a view that has to be refined and transformed to certain platforms and renderings [10]. Besides UsiXML, UIML [8] specifications can be used to define a view composed of abstract elements in its structure section, which will be refined by a peer section to translate the view elements to certain GUI framework components or user interface controls.

Many user interface pattern libraries like [12][13] only focus on the view aspect. Metaphors [4][11] like trash bins and shopping carts may represent the foundation for the views of user interface patterns, but they also drive the aspect of interaction.

Secondly, a user interface pattern embodies a stereotype **interaction**. An interaction between a user and several user interface pattern instances of a certain type is always perceived and performed in the same way by the user. For example, each time a user interacts with a "Search Box" [12], he inputs the search string, selects the search category using the list box and finally triggers the actual search with the button. The options and sequences of interaction along with related behavior are defined independently from the context the pattern is being used in. Another example underlines that: A set of checkboxes is used to select only two options out of many available. The user interface pattern has to enable this constraint in its definition, regardless of the actual number of checkboxes within the possible pattern instances. Forming a unit of general purpose and applicability together with the view aspect, the interaction aspect adds essential value to the user interface pattern definition, which is reusable in many contexts, accordingly. The interaction strongly relies on and refers to the view aspect. This unity of view and interaction primarily forms the reusable entity and distinguishes the user interface pattern from ordinary GUI framework components and composite user interface controls.

Thirdly, besides the first two mandatory aspects, a user interface pattern may define an optional context dependent **control**. This aspect is primarily needed for user interface patterns that are composed of several user interface controls or even other user interface patterns. These composite patterns react on the context they are applied to by selecting, instantiating and configuring their child elements. An example for such a pattern is given by the "Advanced Search" [12], which enables the user to select search criteria depending on the object to be searched. This particular pattern offers "a special function with extended term matching, scoping and output options", when "users need to find a specific item in a large collection of items" [12]. A possible interface of an advanced search pattern instance is drawn in Figure 1.

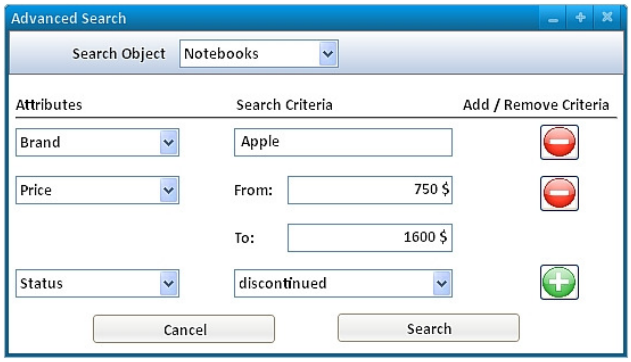

Figure 1. Interface example for an advanced search dialog

Each search criterion line refers to one of the object attributes' data type and thus can be regarded as a smaller user interface pattern that is instantiated on demand. For all money types, as shown in Figure 1, two values can be entered as search parameters. Each time a money type occurs, the same view and behavior are to be instantiated, hence this type of search criterion line is defined as a user interface pattern.

Another example is depicted in Figure 2. This dialog is composed of several user interface patterns working together. A "Data Table" [13], which is configured according to the object to be displayed, is arranged on the right hand side. On the left hand side, a search refinement can be specified using the given criteria, which are derived from the objects' attributes and their data. The main user interface pattern defines the entire "Search Results" tab, configures and instantiates its child patterns depending on the object and its attributes to be searched. Eventually, the interaction aspect of the dialog is distributed along the pattern instances. The controlling aspect of the main pattern handles the lifecycle of each child pattern instance and queries their interaction events in order to complete its own interaction sequence. For example, only the activated search criteria in Figure 2 are considered for compiling search data, when the button "Refine Search" is activated. Thus, the second aspect of user interface patterns provides the input for the controlling mechanism of more sophisticated or hierarchical user interface patterns. The need for a controlling aspect depends on the structure and purpose of the pattern itself. The simple search box does not need the third aspect, since it always features the same visuals, configured data and output events or data. Its behavior is limited to states that can be determined at design time easily. In contrast, the states of the "Advanced Search" or "Search Refinement" can be determined only at runtime, with

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-230-1

knowledge about the application context, and finally, user inputs.

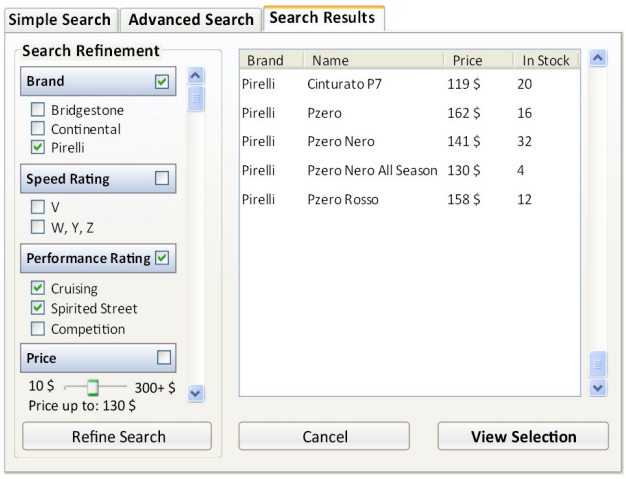

Figure 2. Interface example for a search results dialog

According to the need for the controlling aspect, more examples of user interface patterns are "Wizard" [12] or reusable dialog types like the "Search Refinement" that may act as templates for several dialog instances. The context the controlling aspect of the pattern relies on can be embodied by a static artifact, e.g., an object and its attributes, or dynamic artifact, e.g., a state machine or task model. By referring to the latter, an implicit connection to the dialog controller of the software architecture can be established.

B. Variability Perspective on User Interface Patterns

The instantiation of a user interface pattern for varying contexts will result in implementations of given architectural components that differ in certain aspects. That is why we refer to the common architectural pattern MVC [23] as a perspective for discussing the adaptability, variability and reuse of user interface patterns in different contexts.

Firstly, the easiest way of applying a user interface pattern in various contexts can be established by adapting it to accept a range of **data types** for its defined view aspect elements. For instance, a "Data Table" [13] being part of a user interface pattern view aspect will be instantiated for displaying a variety of business objects with different data types in their attributes. Another example is given by "Event Calendar" [12] or similar patterns, which interpret the given values within the model by proving an appropriate display of data. This kind of reuse would only affect the model part of the architecture.

Secondly, user interface patterns need to be adapted to the actual **dialog layout**. A "Double Tab Navigation" [12] needs to be shaped to the actual menu contents and layout to be displayed, for example. This results in a change of the presentation (view) component and related events, where the number, ordering and layout of required user interface controls have to be determined and implemented accordingly. Consequently, user interface patterns act as templates for the static and dynamic aspects of the view component and its presentation control. Therefore, the view and its related controller have to be adapted.

Thirdly, besides the prior concerns, composite user interface patterns have to feature variability regarding their **controlling aspect**. The controller of an MVC triad can be considered to be acting on two different levels. One part of the controller is responsible for the visual event handling only and is closely related to the view aspect of a user interface pattern instance. Due to cohesion and coupling concerns, the scope of this controller should be limited to one visual design unit, meaning one user interface pattern instance and its specified behavior at a time. The other part of the controller should handle the application related or logical behavior. Since user interface patterns can be composite, controllers should follow the same structure and be assigned to the individual pattern instances. With this compositional structure of the patterns and the controllers accordingly, the reuse of certain combinations of patterns will be facilitated.

An example depicting the variability of user interface patterns is given in Figure 3. On each side of the upper half a visual representation of a user interface pattern specification is shown. The first dialog sketch defines the view used for a business object and the tabs, which establish the navigation structure a user might interact with. The second dialog sketch above visualizes a sub-pattern that is used for the "Properties" tab. Therefore, the example consists of a composite user interface pattern. Possible instances of the two patterns are shown below. Concerning variability of model data and presentation (view), the specific dialog on the lower right hand side shows that displayed data and corresponding user interface controls are chosen dynamically for the object the pattern instance is assigned to.

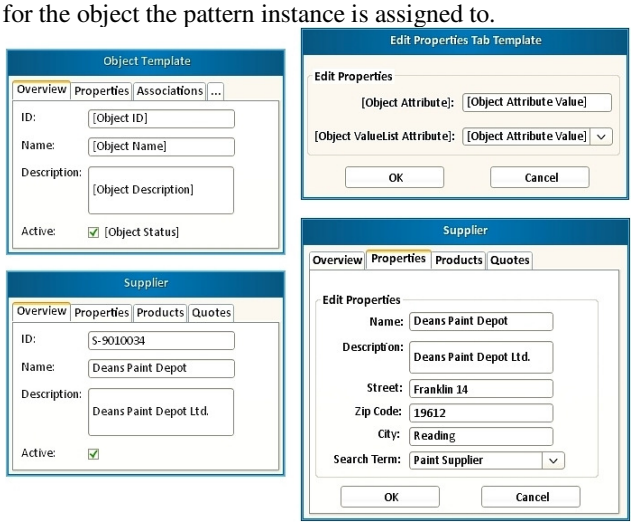

Figure 3. User interface pattern templates (above) and instances (below)

Especially the "Search Term" attribute is to be mentioned, as there is a distinction between text fields and list boxes regarding the data type. The lower left hand side dialog sketch has fixed visuals and data assignments, but it is variable, as it considers the actual type and number of associations an object may possess. For each association, an assignment dialog is presented that can be accessed by the dynamically instantiated tab. In the example, "Products" and "Quotes" tabs refer to the associations of the object "Supplier".

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-230-1

To conclude, a user interface pattern specification has to enable the definition and distinction of all three aspects in order to provide the preconditions for effective reuse and variability. User interface patterns are meant to be adapted to different data types to be displayed. In addition, they need to be aware of the number and layout of their view components. Lastly, user interface patterns do not only need to adapt to their own variable interaction, which depends on actual view component instances, furthermore they need to define a variable control to enable the collaboration of and interaction with their child elements.

C. *Classifying Dimensions for User Interface Patterns*

We refer to the following dimensions to classify user interface patterns:

The **degree of formalization** distinguishes between descriptive and generative patterns [1]. The argument has been raised that a user interface pattern needs a rich human understandable specification. The latter resembles merely a description in prose and represents descriptive patterns, which cannot be processed by generators and other tools of the development environment. Thus, a machine-readable form amends the user interface pattern entity to a generative pattern [1].

The **user paradigm** reflects how the users' tasks will be supported by the entire user interface. Ludolph [4] mentions the design of object-oriented user interfaces, which enable the user to manipulate only one object in a dialog at once, as well as the procedural paradigm, which allows the user to accomplish a complete process consisting of several steps in a defined sequence. These options are complemented by the function-oriented paradigm, which provides a dialog for completing a certain step or complex task out of a process working with more than one object. The user interface patterns vary in their capability to support the three paradigms. For instance, the "Wizard" [12] is intended to build a procedural user interface. Other user interface patterns can be compiled to display the data of several business objects and form a collaboration to support the user concerning a certain function.

The **variability** of the user interface pattern can also serve as a dimension. There are patterns, which hardly feature any variability between their instances. For instance, "Breadcrumbs" [13], an "Event Calendar" [12], or a "Date Selector" [12] always feature the same abstract visuals and interactive behavior. So these patterns are called static or invariant patterns, with respect to the visual and interaction aspects. The other patterns with true variability in view, interaction and even control can be called dynamic user interface patterns.

A final dimension can be proposed with the **application area**: Firstly, user interface patterns can be interface-specific (graphical user interfaces — GUI, text-based interface or spoken dialog systems, etc.). Secondly, paradigm-specific (WIMP or touch-based interface, etc.) patterns can be differed. Thirdly, some system-specific patterns (Windows, MacOS, Android or iOS, etc.) have emerged from the appropriate GUI specification guidelines. Finally, user interface patterns can be closely associated with a certain domain (eBusiness, simulation systems, etc.). Remarkable reuse across different systems in similar use cases of a domain may be driven by a stable set of user interface patterns.

Finally, user interface patterns within the given dimension can be related to each other. For example, interface-specific user interface patterns often do have different system-specific appearances. Particularly for descriptive and generative descriptions of one pattern, we suggest that they should be made available in a linked form in future user interface pattern libraries in order to facilitate the understandability of both human and machine involved in the same development process. In this context, current (descriptive) patterns libraries also have to be checked if all containing pattern descriptions fulfill the aforementioned definition and criteria of a user interface pattern.

IV. FORMAL DESCRIPTION OF GUI PATTERNS

In the following two sections, we describe an assessment of the capabilities of pattern-based user interface development with respect to the application of current methods and notations. To accomplish this, we outline two practical examples of formalizing and utilizing patterns from general description to their application in source code. Since the state of research in generative user interface patterns mainly focuses on the WIMP paradigm, we also concentrate on that area.

GUI patterns are generative user interface patterns with an application area in WIMP software. Formal notations are necessary to implement generative GUI patterns. Since there is no generally used pattern language, independent user interface description languages are widely applied for formalizing GUI patterns (see Section II). In our prior work, we conducted an extensive investigation on formal graphical user interface specification languages and their applicability for GUI patterns. Such languages offer elements like templates (UIML) and abstract as well as concrete models (UsiXML). Both have been developed further by extensive work of research and have reached a high level of maturity. Therefore, we used UIML and UsiXML for the formalization of exemplary GUI patterns. For our analysis, we focus on the GUI pattern "Advanced Search" (Figure 1).

For a formalization of the advanced search pattern in UIML and UsiXML, at first we analyzed the components and dynamics of the pattern and found the following contents:

Advanced search view aspect:

- User interface controls: text field, dropdown list, checkbox, button
- Possible sub-pattern: „Date Selector" [12] for date data types within the objects
- Layout: four-column grid with a dynamically varying number of rows (search attribute, search criterion or value, logical conjunction, add or remove function)

Advanced search interaction aspect:

- Input parameters consisting of attributes and their values of searchable objects
- Output result: logical conjunction of search clauses

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-230-1

Advanced search control aspect:

- 1. Selection of search criteria from dropdown list determines input form of search value,
- 2. Click on plus button adds another search clause,
- 3. Click on minus button deletes last search clause,
- 4. Click on search button sends finished search clauses

These results can be used as specifications for a formal notation of the advanced search pattern. In UIML, a static interface part (view aspect) is described in structure tags, while changes in this part during runtime, which are triggered by user interaction, can be described in behavior tags (Figure 4). By implementing certain rules of changing structural code depending on input, the interface can be manipulated in various ways. These rules contain the condition they are triggered by and the specific action, which is performed. Through the application of parameter-driven templates, parts of structural code, and thus portions of the view aspect, can also be reused. By implementing these UIML concepts, the view and interaction aspects of an advanced search can be represented.

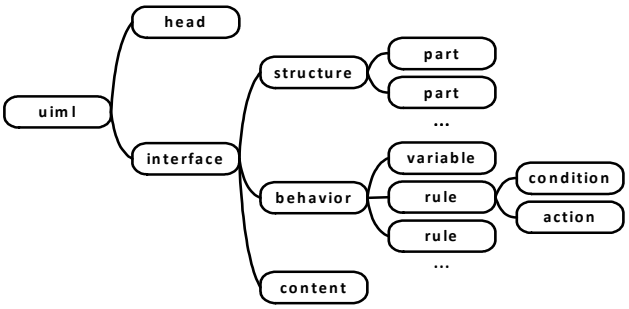

Figure 4. UIML structure for advanced search description

The UsiXML language relies on more complex and methodic specifications. Here, different kinds of models in a model-based interface development process are proposed. The most important are the following: The abstract user interface model (AUI), where a user interface can be described independently from the type of interface, paradigm, system or software (see Section III, application area of user interface patterns). Such abstract models can be concretized in the concrete user interface model (CUI), which relies on GUI description, much like UIML. Other models describe the processes of interaction with the planned interface (task model), static data and functions of it (domain model) or show connections between the different models (mapping model) [10]. For our description of an advanced search GUI pattern we focused on the CUI, where a GUI part can be differentiated into several windows with their own user interface controls and behaviors (Figure 5). However, UsiXML does not allow the use of variables or dynamic manipulations of already described window contents, like UIML does. Therefore, a complete advanced search with a potentially unlimited number of search clauses could not be implemented.

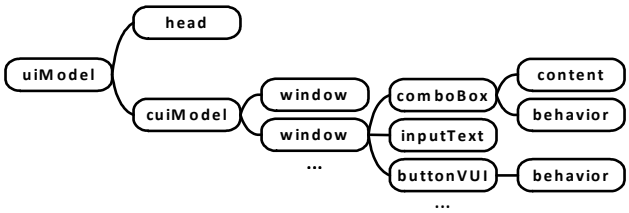

Figure 5. UsiXML structure for advanced search description

While XML is a good format for the view aspect of machine and human readable user interface patterns and therefore, in a way, generative as well as descriptive patterns, major problems of the use of interface description languages arise from the nature of patterns: Those languages are not created for the storage of incomplete, template-like interface descriptions, which are missing all concrete specifications, e.g., of user interface controls. This incomplete description often cannot be fully linked to the interaction or control aspects of the pattern. Certain limitations of the description languages, especially in UsiXML, also prevent the complete implementation of the interaction or control aspect. Furthermore, most of the languages are adapted to graphical user interfaces under the WIMP paradigm and do not allow the description of other interface types (an exception is the UsiXML AUI model). To achieve a full variability, which supports all mentioned aspects and dimensions outlined in Section III, the option to describe other interface types would be necessary. Finally, code in independent user interface description languages is built to be rendered in the user interface programming language, once the development is nearing completion. Here, several renderers for UIML and UsiXML already exist. The integration of user interface patterns into the code generation process, however, is not comparable to a rendering, since these patterns need to be instantiated first. The following subsection deals with a concept for these necessary development steps.

V. INSTANTIATION AND CODE GENERATION

Necessary for the application of existing generative user interface patterns is their procedural and technical integration into user interface development. This includes the steps of identification, selection, instantiation and integration of user interface patterns [3].

An **identification** of patterns in a planned user interface can take place at the modeling stage. The occurrence of user interface patterns can be identified in dynamic descriptions of a desired interaction process, namely in task models, or in static model components, like class diagrams. This part of pattern-based development is well-researched (see Subsection II.B for references to examples).

Also, the **selection** of patterns can be accomplished easily. Formalized and generative user interface patterns have to be stored in a pattern repository. Upon identification of patterns in a model, a list of suggestions with identified patterns should be displayed and desired patterns can be selected. Again, suggestions for pattern storage and selection have been made in the references in Subsection II.B.

Identification and selection of patterns are part of system interface modeling. Thus, the described technical solutions

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-230-1

can be plugins or special applications for the integration of patterns in this process.

The next necessary step is the **instantiation** of patterns. Since software patterns are general descriptions, which are independent from a concrete modeling or implementation scenario, specific details are missing. For this reason, the user interface descriptions outlined in Section IV are incomplete, template-like. For example, in the advanced search pattern, as described above, the content and layout of the dropdown list a user selects attributes of the searchable objects with is missing, since these objects and their attributes vary in each specific implementation of an advanced search (searching in emails, products, pictures, etc.).

The instantiation fills these gaps in a user interface pattern with specific values. Thus, the result of an instantiated user interface pattern is a complete description of this special part of the user interface. For the use of independent interface descriptions, like UIML and UsiXML, that means a complete description and a valid XML-based document is achieved only after instantiation. An instantiated user interface pattern in UIML or UsiXML can be rendered in the final interface language. Therefore, the general process of generating source code from user interface patterns will be as depicted in Figure 6.

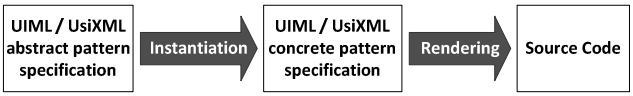

Figure 6. User interface pattern instantiantion and rendering

The next step is the **integration** of instantiated patterns in the interface implementation. Besides rendering, the main task of the integration is the establishment and application of a relation between selected patterns or patterns and other parts of the source code. The key to these relations is the existence of defined input parameters and output results for each pattern. For the advanced search pattern as an example, input parameters and output results are defined in Section IV. The output result of a user interface pattern, e.g., a set of found objects from an advanced search, can serve as an input for another pattern, in this case a search results pattern [12]. Or, input parameters and output results of interactions in user interface patterns can be used to connect the integration of patterns in components of finished source code manually.

For XML-based user interface description languages, renderers can be applied to get source code from instantiated patterns. Since for our example, advanced search, no sufficient renderer was available, we implemented XSLT scripts for the transformation of UIML and UsiXML patterns into JavaScript and HTML code.

VI. RESULTS AND DISCUSSION

Through the exemplary formalization, instantiation and code generation of the user interface pattern “Advanced Search”, we could assess the possibilities and limitations of current methods for pattern-based user interface development. Basically, a formal description of GUI patterns is possible, and after that, they can be instantiated and transformed into source code.

The application of UsiXML shows that, while it supports abstract user interface models, it does not allow dynamic creation and manipulation of interface parts in a UsiXML document yet. Therefore, UIML is better suited to store user interface patterns in an existing XML-based interface description language.

However, user interface description languages are not exactly suitable for the storage of user interface patterns, as shown in the previous sections. They are missing options of template-like interface descriptions without layout or content specifications, so that only after instantiation, valid descriptions are established. Thus, the first of our criteria for the analysis of current pattern-based interface development methods from Section I.B, the variability and reusability of stored patterns, is not met through the use of general XML-based description languages.

A composition of user interface patterns and their integration into the source code is also possible through the steps outlined in Section V. A full composition ability of user interface patterns to form a hierarchy of GUI components, however, fails with current established methods because there is no standardized functionality of pattern storage, instantiation and code integration. Such a part of development tools could be called pattern manager and should be able to suggest, instantiate, connect and generate source code of user interface patterns, which are stored in a pattern repository. A standard exchange format of communication between patterns is also missing, since our definition of input parameters and output results is applicable, but arbitrary and not further developed. Thus, the second criterion of our objectives is also not met.

A pattern instantiation into varying interface paradigms and types as named in our last criterion is not possible with the application of GUI-specific description languages. UsiXML supports an abstract user interface model, but only as a component of a GUI description, not as a separately usable model. The degree of abstraction of the AUI is too high; it does not contain a complete interaction or communication model, so that it is not sufficient for the storage of a complete user interface pattern.

An implementation of variability, hierarchy and interaction of a composition of user interface patterns with the application of current notations is a very difficult task. Moreover, a pattern lifecycle with independent formalization, instantiation and code generation is very extensive and could be less complex.

In our conclusion we will use the developed criteria and found shortcomings of current pattern-based interface development to define first specifications of improved methods and notations.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have shown that current solutions for pattern-based user interface development do not meet the criteria of a complete and efficient design method for any kind of user interface.

Based on the results of our practical formalization, instantiation and code generation of the advanced search user interface pattern, we propose the following specifications of

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-230-1

a sufficient pattern description and development integration method:

- An exact **definition** of generative user interface patterns: inclusion and exclusion criteria, characteristics, adaptability and composition ability should be established to describe them as artifacts in the development process. We proposed a first version in Section III.
- A **metamodel** for the structure and behavior of user interface patterns, which reflects the defined aspects. It would serve as a guideline for a standardized pattern implementation, as well as a method for traceability of certain aspects between different phases of the interface development.
- A specialized **pattern language**, which allows an exact and formal representation of patterns according to the definition and their metamodel.

And, as a practical addition:

- A development tool or module for user interface development, which contains the **pattern repository** and the **pattern manager** and offers the functionality described in Section VI. Here, it should be resorted to implementation and storage standards to assure the availability of such a tool in different development environments. The full lifecycle of user interface patterns, from their creation to their application and further development, should be supported.

Based upon these specifications, a practical solution can be approached. In our further research, we plan to concentrate on proposing a metamodel for generative user interface patterns as well as a first draft for a special description language for user interface patterns.

REFERENCES

[1] 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), ACM, June 2012, pp. 12-19, doi: 10.1145/1824749.1824753.

[2] R. Beale and B. Bordbar, "Pattern Tool Support to Guide Interface Design," Human-Computer Interaction (INTERACT 2011), LNCS Vol. 6947, 2011, pp. 359-375.

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

[4] M. Ludolph, "Model-based User Interface Design: Successive Transformations of a Task/Object Model," in User Interface Design: Bridging the Gap from User Requirements to Design, CRC Press, Boca Raton, Ed.: L.E. Wood, 1998, pp. 81-108.

[5] P. Chlebek, "User Interface-orientierte Softwarearchitektur," Mainz: Vieweg, 2006.

[6] Mozilla Developer Network, "XUL," https://developer. mozilla.org/en/XUL 25.06.2012.

[7] Microsoft, "XAML in WPF," http://msdn.microsoft.com/en-us/library/ms747122.aspx 25.06.2012.

[8] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster, "UIML: An Appliance-Independent XML User Interface Language," Proc. Eighth International World Wide Web Conference (WWW'8), Elsevier Science Pub., May 1999.

[9] UIML 4.0 specification, http://www.oasis-open.org/ committees/tc_home.php?wg_abbrev=uiml 10.05.2012.

[10] J. Vanderdonckt, Q. Limbourg, B. Michotte, L. Bouillon, D. Trevisan, and M. Florins, "UsiXML: a User Interface Description Language for Specifying multimodal User Interfaces," Proc. W3C Workshop on Multimodal Interaction (WMI'2004), 19-20 July 2004.

[11] S. Wendler, D. Ammon, T. Kikova, and I. Philippow, "Development of Graphical User Interfaces based on User Interface Patterns," Proc. PATTERNS 2012, 22-27 July 2012.

[12] M. van Welie, "A pattern library for interaction design," http://www.welie.com 10.05.2012.

[13] Open UI Pattern Library, http://www.patternry.com 10.05.2012.

[14] A. Toxboe, "User Interface Design Pattern Library," http://www.ui-patterns.com 10.05.2012.

[15] J. Engel, C. Herdin, and C. Maertin, "Exploiting HCI Pattern Collections for User Interface Generation," Proc. PATTERNS 2012, 22-27 July 2012.

[16] A. Wolff, P. Forbrig, and D. Reichart, "Tool Support for Model-Based Generation of Advanced User Interfaces," Proc. MoDELS'05 Workshop on Model Driven Development of Advanced User Interfaces, Montego Bay, Jamaica, 2 October 2005.

[17] A. Wolff, P. Forbrig, A. Dittmar, and D. Reichart, "Tool Support for an Evolutionary Design Process using Patterns," Proc. Workshop on Multi-channel Adaptive Context-sensitive (MAC) Systems: Building Links Between Research Communities, Glasgow, 15 May 2006.

[18] M. Wurdel, P. Forbrig, T. Radhakrishnan, and D. Sinnig, "Patterns for Task- and Dialog-Modeling," J.A. Jacko (ed.) HCI International 2007, Beijing, 22-27 July 2007, pp. 1226-1235.

[19] D. Reichart, A. Dittmar, P. Forbrig, and M. Wurdel, "Tool Support for Representing Task Models, Dialog Models and User-Interface Specifications," Interactive Systems, Design, Specification, and Verification (DSVIS'2008), LNCS Vol. 4323, 2008, pp. 92-95.

[20] A. Wolff and P. Forbrig, "Deriving User Interfaces from Task Models," Proc. the 4th International Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI 2009). Sanibel Island, USA, 8 February 2009.

[21] S. Fincher, "PLML: Pattern Language Markup Language," http://www.cs.kent.ac.uk/people/staff/saf/patterns/plml.html 25.06.2012

[22] F. Radeke and P. Forbrig, "Patterns in Task-based Modeling of User Interfaces," M. Winckler, H. Johnson, P. Palanque (Eds.): Proc. 6th International Workshop on Task Models and Diagrams for User Interface Design (TAMODIA'07), Toulouse, France, 7-9 November 2007.

[23] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stahl, A System of Patterns, New York: Wiley, 1996.

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-230-1