Requirements for a Definition of Generative User Interface Patterns

Stefan Wendler and Ilka Philippow

Ilmenau University of Technology, Software Systems, Process Informatics Department
Helmholtzplatz 5, 98693 Ilmenau, Germany
{stefan.wendler,ilka.philippow}@tu-ilmenau.de

Abstract. Patterns for visual GUI design propagate the specification of user interfaces with proven usability and motivate model-based development processes with increased reuse of GUI component compositions. However, a common structure, that captures all the reusability and variability demands, neither has been established for the descriptive form nor the generative kind of user interface patterns. Dedicated GUI specification languages like UIML and UsiXML fail to express pattern definitions that can be instantiated in varying contexts. Thus, model-based processes are required to introduce own media to store those patterns. With our approach, we review the state of the art for generative user interface pattern definition and derive requirements which we refine by a Global Analysis. Finally, we developed a model that accommodates primary factors and their impacts towards the concept for a more sophisticated generative user interface pattern definition.

Keywords: HCI patterns, user interface patterns, GUI generation.

1 Introduction

For systems which are intended to provide a direct support for users in their operative tasks the user interface is of highest importance. The developers of the user interface have to be aware of three different basic requirements. Firstly, the user interface has to provide an effective and task-adequate access to the functional layer and data of a given system. Moreover, the user interface has to be visually designed and implemented in a way that enables the user to work with the system efficiently. Lastly, a business information system has to meet the before mentioned requirements after incremental adaptations to new demands imposed by the changed business processes and their environment.

Finally, these requirements for user interface systems lead to high efforts in initial development and the further lifecycle of the system. During the adaptation of a user interface to user requirements some aspects of the presentation layer essentials may see a potential for increased reuse. For these aspects, the basic layout of dialog types, their arrangement and navigation mechanism as well as reoccurring user interface controls (UI-controls) and their data type processing are considered to see more reuse in the future. System specific patterns seem to be helpful for the reuse as these aspects

M. Kurosu (Ed.): Human-Computer Interaction, Part I, HCII 2013, LNCS 8004, pp. 510–520, 2013.

feature a high variability, e.g. the content data of dialogs as well as the associated UI-controls and navigation options highly depend on the task to be supported. Finally, a need has emerged to both decrease the efforts for GUI development or individual customization and enable a homogenous assembly for of the architecture comprising the user interface components.

HCI-Patterns. The deployment of patterns for GUI development has been discussed for more than a decade now [1][2][3]. Besides architectural patterns that can be relied on for the definition of basic structures for GUI components, various definitions, approaches and modeling process have emerged from the application of patterns that provide solutions for the visual and interactive parts of the GUI, which are not addressed by the basic design patterns. However, no consistent definition for patterns dedicated to GUI development has been established yet.

In general, as increased reuse is propagated here and by other approaches applying model-driven processes [4], there is a need for a dedicated pattern definition. The pattern conception has to ensure that a GUI system will be developed homogenously along its hierarchy of visual and non-visual components, meaning that architecture and GUI patterns have to be comprehensive. If the pattern concept was not able to cover every context of application and thus in need for specialized solutions, the main problem of GUI development related efforts would persist. We claim that this is an issue for current research projects that may enable pattern-based solutions for GUI systems but at some point have to revert to manually refined structures not covered by mere pattern instantiation.

User Interface Patterns. To begin with, "user interface pattern" (UIP) [5] is used as a term for the further discussion of GUI specific patterns. UIPs are intended to aid in adaptation and creation of user interfaces with a similarity in task or data processing, visual and interaction design. Currently, UIPs are not considered as a strong asset as architectural or design patterns for enabling reuse of concepts and context aware instantiation. It is our goal to encourage a basic conceptual view on UIPs that may pave the way for their assured and unified integration as an artifact in development environments. Moreover, we strive for the elaboration of a requirement model for UIPs that should be able to capture all essential aspects and needs for context-specific application and instantiation of UIPs. GUIs created by generators tend to lack usability what could be improved by the involvement of established UIPs [6]. In sum, a process that enables the instantiation of UIPs and their composition to form a user interface of high usability and adaptability altogether would be of great value.

Objectives. As our objective, we see the review of the state of the art in the area of implementation related UIP approaches. After a problem analysis concerning the formalization of abstract UIPs, we formulate requirements that reflect the exploits of UIPs. During the analysis of requirements, we derive influence factors that are systematically presented via a Global Analysis.

Our aim is to raise the awareness of expert groups to focus on UIPs and their abstract pattern nature. The purpose behind UIPs was to act as patterns describing

user interface commonalities and allowing instantiation. A first step for the formalization of UIPs required for automated processing is the identification of their characteristics and traits which embody the reusable aspects of a user interface.

2 State of the Art Review

2.1 UIP Definition and Collections

Stating a UIP definition is not easily done since a standardization of this term has yet to be reached [7]. What can be assumed is a separation of UIPs between their usage in specification or generation of software, hence the pattern idea for GUI systems found its roots in HCI [8] and now evolves towards automated or generative development [9]. As a result, UIPs have been separated into two types by Vanderdonckt:

Descriptive UIPs. Serving mainly as illustrative examples for GUI specification, descriptive UIPs are described and interpreted by human-beings and thus act as inspirations or best-practices for usability proven GUI design.

Generative UIPs. For our objective, the generative type of UIPs is of greater importance since these patterns have to store all relevant information for the automated processing by generators. For a set of given generative UIPs a defined set of design patterns or architectural structures have to be instantiated in source code. Each type of UIP determines a certain quality of architecture part. The final choice of UIP instances to be used for a part of GUI system merely determines the quantity of code structures to be instantiated by the generator. As reuse should be increased, the manual addition of linking code (glue code) must be omitted.

PLML. The descriptive specification of UIPs already poses an issue which is caused by the lack of a proper definition for this artifact. There have been attempts to create a unified description scheme with the introduction of PLML (pattern language markup language) in 2003 [10]. Engel, Herdin and Märtin conducted an investigation of common description schemes in [7]. Descriptive UIPs are established as specification elements and supported by the HCI community. Thus, one could be tempted to abstract the common structure of UIPs from the existing UIP collections. Following this course, Engel, Herdin and Märtin have discovered that a full compilation of necessary elements is difficult, since the patterns are still presented in a rather unstructured form [7] and missing attributes for technical considerations. In addition, PLML never was supported by a unified metamodel that satisfies all generative UIP needs. Several extensions have been proposed but there are still enhancements which have to be incorporated [7]. Furthermore, implementation aspects and relationships among UIPs are neither sufficiently nor clearly mentioned by PLML. With this status quo on the specification level, a proper formalization of UIPs as generative artifacts is hindered at an early stage.

UIP-Libraries. To some extent, descriptive UIPs have been filed in UIP-libraries like [11] and [12]. In contrast to the weaknesses of their content structure, UIP-libraries do motivate our approach towards a clearer definition of generative UIPs. They drive the pattern-based application of UIPs by depicting GUI example layouts, that do feature stereotype visuals and interactions which may be adapted to individual application contexts. Thus, UIP-libraries inspire the idea to compose individual GUI dialogs by choosing from the available pattern palette.

2.2 Modeling Processes Involving UIPs

A sophisticated development environment for generative UIPs was created by the University of Rostock [4]. Their example proves that UIPs can be instantiated to various application contexts and thus facilitate reuse in GUI development. However, the used presentation model implicitly defined the UIP to be applied. Consequently, a selection of a different UIP had to be done via a manual replacement by pre-defined GUI components (PICs) already resembling certain instances of UIPs. Thus, the variability was restricted to few applications. Besides, not all types of UIPs were supported or manual adjustments still were needed. Later on, a vote for the closer integration of model- and pattern-based processes was raised [13]. This goal implicitly demands for a mature definition of generative UIPs as there was still potential to increase reuse and lessen efforts for linking and integration models to be generated. In this context, UIPs were to be stored in an abstract form so that they could be instantiated. Finally, options for their formalization had to be considered.

Following this idea, modeling frameworks and processes for GUIs [14][15][16] are advancing and have already introduced their own notations for generative UIPs. However, these approaches are difficult to judge, since they are mostly presented as drafts and miss profound code examples. In fact, there exists no common requirement model for the common or similar goals they are striving to achieve. Their individual notations for UIPs are either based on customizations [14][16] of available GUI specification languages or even own XML language conceptions [15].

2.3 UIP Formalization

GUI Architecture Concerns. To assess the applicability of GUI specification languages for generative UIPs, feasible criteria have to be sourced. As the requirements and responsibilities for a component-based GUI system [17] alone are extensive, the patterns, which are intended to drive the creation of those components in a generative way, also have to be capable of supporting several concerns at once. Due to the fact that architectural patterns like MVC, PAC or the Quasar reference architecture for clients [18] are needed as additional mental models framing the rather elementary and universal structures of classes and components, the target architecture is of a complex basis, which rarely has been unified in its details. Without this unity of a common architectural basis and the dilemma concerning reference architectures [18], a formal UIP definition faces the problem to be acceptable for different architectural pattern interpretations and implementations.

Criteria. To avoid those architecture related issues, we first set up three fundamental criteria to be met by formal UIPs and came up with a generalized reflection of the variability concerns of UIPs by referring to a simplified MVC model covering basic responsibilities needed in most applications [19]. The interrelation of the criteria and variability demand for UIPs that can be combined freely and integrated in GUI entities without manual design modeling, as it would be needed for common architectural and design patterns. Formal UIPs have to enable at least the two criteria "reusability and variability of stored user interface patterns" and "ability of user interface patterns to be composed in order to form a hierarchy of GUI components" out of their pure pattern form. The latter is a main issue when UIP instances have to be created from their formal XML specifications.

Formal Languages for GUI Specification. In our previous work [5][19], we already went into the possibilities to express generative UIPs with the means of mature GUI specification languages UIML (User Interface Markup Language) [20] and UsiXML (USer Interface eXtensible Markup Language) [21]. Although these XML languages are focused on platform-independent GUI specification and intended to be machine processed, our assessment of UIML and UsiXML revealed that both languages failed in architecture and specification experiments to fully express UIPs with the two considered criteria. We assume that the languages are sophisticated tools for GUI specification and may be used as external domain specific languages for GUI generation, but they are not based on abstract patterns and do lack a conceptual definition of UIPs. However, developers have to revert to existing GUI specification languages, as there is still no dedicated language for UIP formalization.

UsiXML. The abstract user interface model (AUI) suggested by UsiXML sounds promising for storing UIPs. However, the facets and abstract interaction objects [9] used as elements therein create a model that is way too abstract to express the elements of specific UIPs, as their general types of UI-controls are mostly known and thus definable. In contrast, the concrete user interface model (CUI) of UsiXML can express platform specific instances of the AUI model contents, e.g., how an input or output facet is structured by certain UI-controls. In this respect, the CUI acts as a direct instantiation of the AUI and no longer resembles a pattern as the visual structure and behavior cannot be parameterized or reused for other contexts. Finally, both models are not suitable for storing UIPs as their abstraction level is not appropriate.

UIML. The UIML language also offers promising features for UIP formalization. With the <structure> section a "virtual tree" [20] of UI-controls is arranged. This tree can be sourced from more than one UIML file at once. In contrast, UsiXML models are stored in a single file. The UIML "virtual tree" can be modified by sub-sections or even other UIML files as they may restructure given parts (repeat, delete, replace and merge sub-trees) of the main tree [20]. Templates and their variables can be applied to adapt reoccurring UIML tree parts to various GUI descriptions. With those features, it is possible to assemble a GUI virtual tree by integration of several UIML files under utilizing most restructuring options. The style of UI-controls can also be governed by

a global UIML definition to ensure a uniform presentation look. Nevertheless, the UIML mechanisms always need concrete inputs for the elements to be processed. For instance, template parameters of UIPs being sourced by other UIPs forming a composite pattern specification have to be specified with constant data like the number of elements to be included. As the UIML file is being specified, the developer has to provide certain input to govern the occurrences of UIML elements or templates. Therefore, the effect of a pattern featuring structural variability is neglected for UIP compositions. In addition, UIML provides no facilities to describe behavior for elements that are abstract and yet to appear when the UIP is instantiated. In the end, UIML specifications will tend to be too concrete to store the abstract UIPs.

To conclude, the main disadvantage of both languages lies in the incapability to provide a separation of UIP definition and instantiation. UIPs need to be specified in a concrete manner in order to be compatible with the schema definitions of the languages. Invariant UIPs like “Date Selector” [11] or “Input Prompt” [12] can be specified by the languages as there is no need for variability. Concerning UIML, elementary UIPs may be expressed by using templates along with parameters, but nested UIPs pose a problem as included UIPs have to resemble a specific instance.

3 Our Approach

Requirements. A first step towards a more sophisticated UIP model definition was the elicitation of requirements. To source the appropriate information, we relied on our previous work [5][19], an industry project in the E-Commerce domain and the presented state of the art. Since the requirements were scattered across concerns and could not be mapped easily to artifacts or rationale, we decided to apply the Global Analysis [22] as a method to create another view on the requirements so that they could be analyzed concerning their impact, relations and strategies.

Global Analysis. The Global Analysis originally serves as a method to systematically derive and describe the leading factors for architecture design. With this analysis, the given set of requirements is assessed concerning the impact of individual requirements on the system design. Requirements with significant impact are marked as factors, which are classified to one of three factor types. For a set of factor impacts, design strategies are elaborated to realize the specific requirements or overcome their restrictions. The method provides steps to relate requirements to certain decisions and high level system artifacts that drive the design of multiple system components. Following this consideration, it is attempted to limit the impact of factors to artifacts or system structures which can be handled more easily.

In our application of the method, we incorporated some adjustments differing from the original source. In short, factors may be detailed as they are composed of nested factors, they may be operationalized by other factors when they cannot be associated to impacts clearly, and finally, the design rationale is incorporated as an additional artifact influenced by the design strategies. An overview of the method we applied for requirement elicitation is provided in Fig. 1, which also serves as a legend.

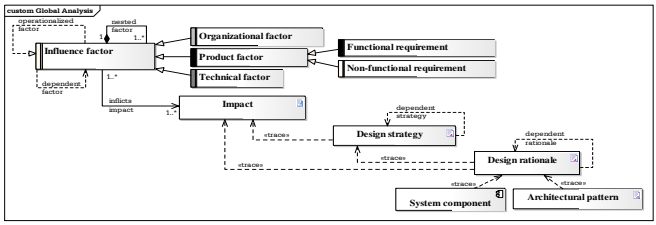

Fig. 1. Metamodel of our customized Global Analysis method

4 Global Analysis Results

The Global Analysis we conducted resulted in the factors presented in Fig. 2. Most factors were derived from the industry project. Consequently, the essential model [23] of the E-Commerce software was to be included as an important factor, since the UIPs should be reused to support existing tasks, functions and objects of that domain. As the user interface needs to be related to a user model [23] and thereby to the essential model, UIPs may be promising, as they may replace the need for a dedicated user model [5]. Hence, UIPs have to be mapped to the essential model. Concerning this matter, extensive work has been conducted by the University of Rostock [4].

The technical factors are based on our consideration to apply XML languages for UIPs already instantiated and thus the description of a concrete GUI [19]. Therefore, the generation of XML code was chosen as an approach which requires a detailed definition of UIPs as shown in Fig 2.

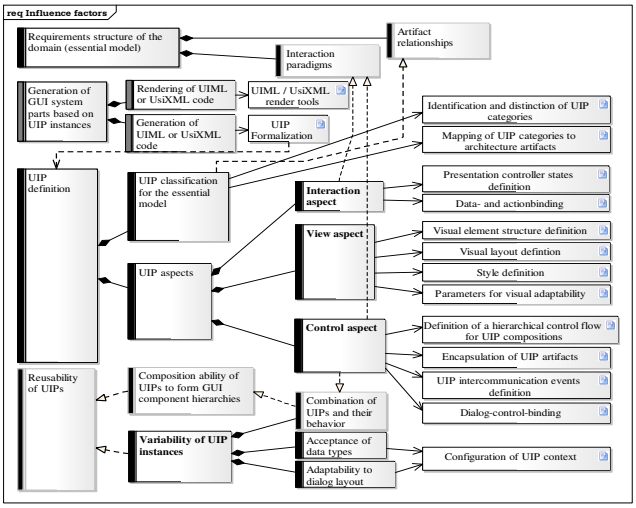

Fig. 2. Influence factors elaborated for the UIP analysis

The UIP definition factor was based on our previous work, as we enriched the three UIP aspects described in [19] with impacts, which provide a more detailed description of their influence on UIP specification models. Concerning the view factor, the impacts are mostly apparent. A view structure has to be defined, along with the layout and style information as the foundation of each UIP. Additionally, parameters have to be considered for the former to enable the adaptation to various contexts.

Being determined by user input the interaction aspect demands for the binding of view structure elements to certain data and presentation actions. The latter may trigger a change of view structure states, e.g., manipulating single UI-controls or interfering with multiple UIPs within hierarchical view structures and their lifecycle.

The control aspect poses the most demanding impacts. For each UIP of a certain class, a corresponding control flow on the same abstraction level has to be defined. In order to allow both the collaboration of composite UIPs and their versatile combination, the encapsulation of UIPs is necessary as well as the communication via defined events. Embedded UIPs are supposed to send events to their controlling UIPs and the latter require to communicate with the dialog controller, which governs the application related states and data.

The reusability factor was derived from the two criteria in Section 2.3. Two other factors, composition ability and variability, were nested in this factor. The former was mainly operationalized by the control factor which already detailed most of the necessary impacts. Initially, the primary requirements of the project were of non-functional kind and have been used to underline the benefits of UIPs. In Fig. 3 the operationalization of these factors is shown.

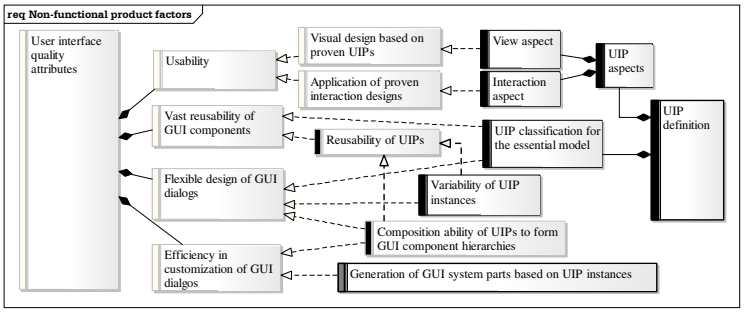

Fig. 3. Operationalized non-functional requirements

Actually, the UIPs were introduced to realize the main projects needs. As considerable HCI specification units, UIPs were supposed to ensure a high degree of usability. Important and at the same time difficult to master requirements on the left hand side in Fig. 3 should be realized by the composition of reusable and generative UIPs.

Design Strategy. The result of the Global Analysis emerged as a solution strategy which demands for the design of a basic UIP description model. As another

requirement model closer to a formalization artifact, it should be able to capture all mandatory characteristics of generative UIPs within a structure. For our objective, we need a general model that represents the requirements more clearly and structured in model whose elements and relationships can be transferred to a formal UIP modeling approach. In this regard, the model has to be independent from GUI specification language definitions of UIML or UsiXML, hence it should be used to overcome the limitations of the former concerning UIP expression. Consequently, the primary concern is to capture the requirements for formal UIPs in a model that may share structural analogies with future formalizations. Furthermore, generative solutions for UIPs are rather seldom and cannot be sourced for our objective. Therefore, the description model should serve as a further enhancement in the assessment of formalization options for generative UIPs and their properties. Finally, it should prove to be applicable in different domains rather than just for one specific application infrastructure.

5 Conclusion and Future Work

With our contribution, a model consisting of influence factors and their impacts related to UIP instantiation, reuse and variability is presented. In the first place, this model can be used to judge to capabilities of available formal languages to express the defined characteristics of a UIP. Besides the judgment, the model can propose applicable enhancements to used languages and models in an environment based on UIPs. In this context, the model also may serve for verification.

The remaining problem is embodied by the need to find an abstract formal representation of generative UIPs. Deploying this form of UIPs, developers will be able to create instances of the same UIP but for a multitude of applications or variations. To establish a new representation of generative UIPs, the final outcome of our Global Analysis suggests to further detail the gathered factors and their impacts with a structural analysis model. The model should fill a gap, where there has not been presented a general model, which describes the structure, elements and relationships of UIPs, yet. So the analysis model will have to consider the following questions:

- How abstract UIPs and their concrete instances can be separately defined?
- Which instantiation parameters are to be defined in abstract UIP specifications?
- How UIP compositions are to be compiled and interaction events are defined in the resulting GUI element cascade?

In any case, a new language or extensions for the XML languages are to be sought after. The new media must facilitate the expression of the generative UIPs, their variability and composition options. For that purpose, a unified UIP model has to be established, which holds all information for the definition of generative UIPs in an abstract form, augmented with parameters allowing for their transformation to single instances or compositions forming a concrete GUI model. To progress towards this goal, existing approaches like [14][15][16] have to be analyzed in detail in order to enhance the presented factor model and discover limitations yet to overcome.

References

1. Mahemoff, M., Johnston, L.: Principles for a usability-oriented pattern language. In: OZCHI 1998, Adelaide, Australia, pp. 132–139 (1998)
2. Todd, E., Kemp, E., Phillips, C.: What makes a good User Interface pattern language? In: AUIC 2004, Dunedin, New Zealand, pp. 91–100 (2004)
3. Dearden, A., Finlay, J.: Pattern Languages in HCI: A critical Review. In: Human-Computer Interaction, vol. 21(1), pp. 49–102 (2006)
4. Wolff, A., Forbrig, P., Dittmar, A., Reichart, D.: Tool Support for an Evolutionary Design Process using Patterns. In: Workshop: Multi-Channel Adaptive Context-sensitive Systems: Building Links Between Research Communities, Glasgow, Scotland, pp. 71–80 (2006)
5. Wendler, S., Ammon, D., Kikova, T., Philippow, I.: Development of Graphical User Interfaces based on User Interface Patterns. In: Proceedings of the 4th International Conferences on Pervasive Patterns and Applications, Nice, France. IARIA Proceedings, pp. 57–66 (2012)
6. Meixner, G., Paterno, F., Vanderdonckt, J.: Past, Present, and Future of Model-Based User Interface Development. i-com 10(3), 2–11 (2011)
7. Engel, J., Herdin, C., Maertin, C.: Exploiting HCI Pattern Collections for User Interface Generation. In: Proceedings of the 4th International Conferences on Pervasive Patterns and Applications, Nice, France. IARIA Proceedings, pp. 36–44 (2012)
8. van Welie, M., van der Veer, G., Eliëns, A.: Patterns as Tools for User Interface Design. In: Farenc, C., Vanderdonckt, J. (eds.) Tools for Working with Guidelines, pp. 313–324. Springer, London (2000)
9. Vanderdonckt, J., Simarro, F.M.: Generative pattern-based Design of User Interfaces. In: Proceedings of the 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems, Berlin, Germany, pp. 12–19 (2010)
10. Fincher, S.: PLML: Pattern Language Markup Language, `http://www.cs.kent.ac.uk/people/staff/saf/patterns/plml.html`
11. van Welie, M.: A pattern library for interaction design, `http://www.welie.com`
12. Open UI Pattern Library, `http://www.patternry.com`
13. Radeke, F., Forbrig, P., Seffah, A., Sinnig, D.: PIM Tool: Support for Pattern-driven and Model-based UI development. In: Coninx, K., Luyten, K., Schneider, K.A. (eds.) TAMODIA 2006. LNCS, vol. 4385, pp. 82–96. Springer, Heidelberg (2007)
14. Radeke, F., Forbrig, P.: Patterns in Task-based Modeling of User Interfaces. In: Winckler, M., Johnson, H. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 184–197. Springer, Heidelberg (2007)
15. Engel, J., Märtin, C.: PaMGIS: A Framework for Pattern-Based Modeling and Generation of Interactive Systems. In: Jacko, J.A. (ed.) Human-Computer Interaction, Part I, HCII 2009. LNCS, vol. 5610, pp. 826–835. Springer, Heidelberg (2009)
16. Seissler, M., Breiner, K., Meixner, G.: Towards Pattern-Driven Engineering of Run-Time Adaptive User Interfaces for Smart Production Environments. In: Jacko, J.A. (ed.) Human-Computer Interaction, Part I, HCII 2011. LNCS, vol. 6761, pp. 299–308. Springer, Heidelberg (2011)
17. Haft, M., Olleck, B.: Komponentenbasierte Client-Architektur. Informatik Spektrum 30(3), 143–158 (2007)
18. Haft, M., Humm, B., Siedersleben, J.: The Architect's Dilemma – Will Reference Architectures Help? In: Reussner, R., Mayer, J., Stafford, J.A., Overhage, S., Becker, S., Schroeder, P.J. (eds.) QoSA 2005 and SOQUA 2005. LNCS, vol. 3712, pp. 106–122. Springer, Heidelberg (2005)

19. Ammon, D., Wendler, S., Kikova, T., Philippow, I.: Specification of Formalized Software Patterns for the Development of User Interfaces. In: The 7th International Conference on Software Engineering Advances, Lisbon, Portugal. IARIA Proceedings, pp. 296–303 (2012)
20. UIML 4.0 specification, http://docs.oasis-open.org/uiml/v4.0/uiml-4.0.html
21. Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M.: UsiXML: a User Interface Description Language for Specifying multimodal User Interfaces. In: WMI 2004, Sophia Antipolis, France, pp. 35–42 (2004)
22. Hofmeister, C., Nord, R., Soni, D.: Applied Software Architecture. Addison-Wesley, Boston (2000)
23. Ludolph, M.: Model-based User Interface Design: Successive Transformations of a Task/Object Model. In: Wood, L.E. (ed.) User Interface Design: Bridging the Gap from User Requirements to Design, pp. 81–108. CRC Press, Boca Raton (1998)