Publiziert in: Alt et al. (Hrsg.), Tagungsband 15. Interuniversitäres Doktorandenseminar Wirtschaftsinformatik der Universitäten Chemnitz, Dresden, Freiberg, Halle-Wittenberg, Jena und Leipzig, Leipzig, 2011, S.64-71.

Generating Graphical User Interfaces for Software Product Lines: A Constraint-based Approach

Johannes Müller

Universität Leipzig, Grimmaische Straße 12, 04109 Leipzig

jmueller@wifa.uni-leipzig.de

Abstract: Due to a high competitive pressure on the global software market, in many areas the software industry is moving from hand crafting to semi-automatic or automatic software construction based on Software Product Lines (SPL). Techniques to automate the construction of software products from SPLs are widely available. These can handle variability in source code artifacts but they are inappropriate to handle variability in Graphical User Interfaces (GUIs). The main reason is that they are not designed to handle such fine grained configurations as they are required to configure GUI frameworks or toolkits. To nevertheless employ them in GUI generation tasks is complex and time consuming. However, in the Human Computer Interaction (HCI) community approaches to develop GUIs in a model-based manner and with constraint-based techniques are worked on that help automate the construction of GUIs. Therefore, the main hypothesis of the proposed research is that constraint-based GUIs techniques are a well suited basis for reducing the customization effort of generated GUIs of SPLs. The paper proposes a research plan to employ these new HCI techniques in generating GUIs for SPLs.

1 Introduction

In many areas software industry moves from hand crafting to semi-automatic or even automatic software construction. These efforts are subsumed under the term *Software Product Line Engineering* (SPLE) where software products that share some features are built on the basis of a common reuse infrastructure. This infrastructure comprises reusable assets such as implementation components, models, generators and other software related artifacts, which have to handle variability of related software products adequately. With SPLE it is possible to reduce development effort, reduce time to market, and increase software quality. Handling variability in implementation components is widely regarded in current research on SPLs and first techniques are developed to handle variability in GUIs as well. However, with todays techniques deriving GUIs from a reuse infrastructure requires in most cases complex customization efforts. Furthermore, these customizations imply manually implemented artifacts for each derived product, which have to be maintained over the whole life cycle of the derived products. An increased development and maintenance effort and hence a reduced efficiency of SPLE is the result. This can render the whole SPLE approach uneconomical.

However, in the Human Computer Interaction (HCI) community approaches to develop GUIs in a model-based manner are worked on that automate the construction of GUIs. Unfortunately, they are designed for single system development. In their groundbreaking work [Pleuss et al. 2010] extend these approaches to SPLs. In their approach GUIs are generated semi-automatically with customizable stub implementations of GUIs.

Since GUI implementations with ordinary frameworks and toolkits are usually inflexible, it is quite likely that the generated implementations have to be customized and hence most of the advantages of SPLs get lost.

If a GUI contains knowledge about usability and design constraints, then it could adapt itself to comply to these requirements. In the field of constraint-based GUIs self-aware GUIs have been developed. A prominent instance is the Auckland Layout Model (ALM) [Lutteroth et al. 2008]. Utilizing such techniques for automatic GUI construction in SPLs would better enable the economic advantages of SPLs also for GUI-based systems.

2 Literature Review

The usability of GUIs is one of the topics of the HCI community. Thus, besides evaluating research into SPLs, HCI research is of particular interest in the next section.

On the one side, in SPL research, a strong body of knowledge has been compiled on automatic construction and synthesis of implementation components in recent years. Two prominent paradigms to automate software production are *Generative Software Development* (GSD) [Czarnecki/Eisenecker 2000]; [Czarnecki 2005] and *Feature Oriented Software Development* (FOSD) [Apel/Kästner 2009]. However, the automatic configuration and synthesis of GUIs in SPLs is only marginally considered so far. Beside some case studies of SPLs that tackle the problem of GUI configuration and synthesis e.g. [Ardis et al. 2000]; [Oliveira 2008] the groundbreaking work of [Pleuss et al. 2010] is the only one that presents a conceptual framework for the configuration and synthesis of GUIs in SPLE. They utilize results of the HCI research stream of *Model Based User Interface Design* (MBUIDs). In a nutshell, they describe the GUI with a task model and some support models in an abstract way. Depending on the selected features, the GUI of the specified product has to fulfill different tasks. Hence, the task model of the GUI of the specified product corresponds to the required tasks. By means of model transformations the task model and the support models are transformed to GUI implementation stubs. However, in most cases these stubs do not meet usability and design requirements and hence have to be customized.

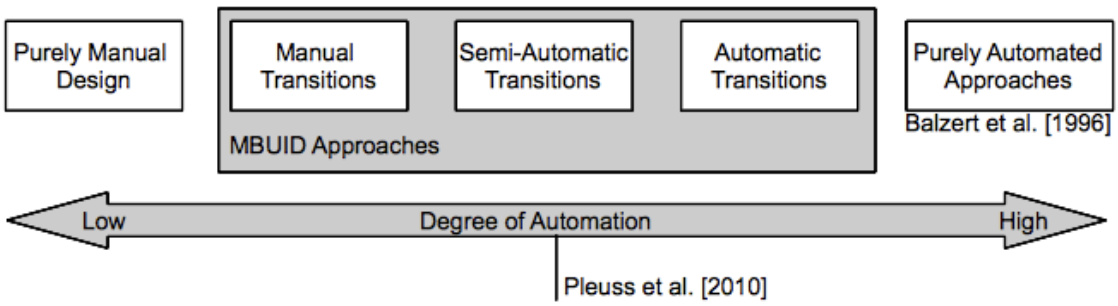

Figure 3: Spectrum of approaches for GUI construction with respect to automation (cf. [Pleuss et al. 2010, p. 71])

On the other side, the HCI community develops techniques to ensure the usability of GUIs. Beside the techniques of MBUID, techniques to automatically or semi-automatically generate GUIs are developed [Balzert et al. 1996]; [Bergman et al. 2002]; [Paterno/Santoro 2002]. Figure 1 depicts the spectrum of approaches for GUI construction with respect to their degree of automation.

GUIs generated with these approaches are only sufficient in very special cases. Mostly they require the software engineer to manually tweak the generated code [Myers 2000] resulting in the same problems mentioned for the approach of [Pleuss et al. 2010].

In another research stream of HCI, namely *Constrained Based Graphical User Interfaces* (CBGUIs), methods and techniques are developed to create self-aware GUIs. They allow to formulate constraints among the widgets that have to be satisfied. Hence, widgets do not need to be placed absolutely or relatively on a window. The actual appearance of a GUI of a single system is the result of a numerical optimization. A prominent approach in this field is the *Auckland Layout Model* (ALM) [Lutteroth et al. 2008]. In a sense, with the constraints the GUI becomes self-aware and can adapt itself to new widget configurations. These techniques could improve the conceptual framework of [Pleuss et al. 2010] and better preserve the advantages of SPLE for GUI-based SPLs.

3 Research Question and Aim of the Proposed Work

As the literature review indicates, the question is whether and how it is possible to employ constraint-based techniques in automating the configuration and synthesis of GUIs for SPLs. The aim is to develop an approach to GUI construction for SPL that moves the approach of [Pleuss et al. 2010] in Figure 1 more to the right. The central research question of the proposed research can be operationalized with the following four specific questions:

(1) What are suitable constraint-based techniques to improve the configuration and synthesis of GUIs for SPLs?

(2) How to adapt them to the configuration and synthesis of GUIs for SPLs?

(3) How to implement a generator on the basis of the conceptual framework of [Pleuss et al. 2010] that utilizes constraint-based techniques?

(4) Is the developed generator capable of generating GUIs that satisfy usability and design requirements?

4 Some Background

As the research question implies, the proposed work will heavily rely on the previous work of [Pleuss et al. 2010] and [Lutteroth et al. 2008],. [Pleuss et al. 2010] provide the surrounding framework, whereas [Lutteroth et al. 2008] provide a promising technique to improve Pleuss' et al. work. To comprehend the general idea of the proposed work, we will give some details to both in the next sections.

4.1 Generating GUIs for SPLs

[Pleuss et al. 2010] identify the problem that current approaches to automate the generation of GUIs often produce less usable GUIs. Hence, they propose an approach that divides the implementation of the GUI into two parts. One that covers the functional part of the GUI, e.g. wiring model code with UI elements or defining the general skeleton of the GUI implementation. The other part covers the configuration of the appearance of the GUI. They utilize MBUID techniques to realize the separation of both concerns. In MBUID the GUI is generated by means of models. As in Model driven Software Engineering, the models are concretized from more abstract representations to

concrete ones, which are finally translated to code. Figure 2 depicts the general MBUID process.

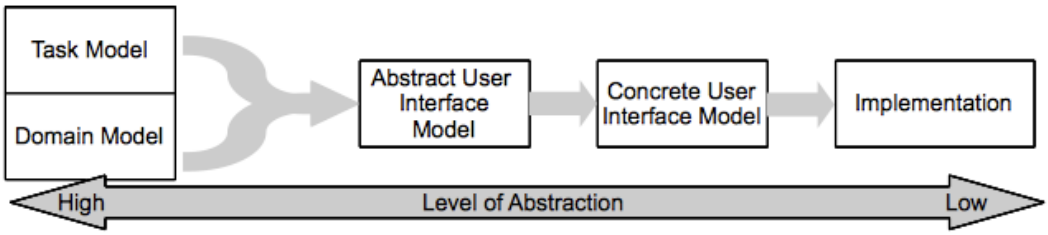

Figure 4: General MBUID Process (cf. [Pleuss et al. 2010, p. 71])

In MBUID the most abstract models are the *Task Model* and the *Domain Model*. The Task Model is an abstract representation of the tasks that are covered by an application. A Task Model covers different types of tasks. An application task is fulfilled by the system, e.g. displaying a value. An interaction task affects the system as well as the user, e.g. typing a value. The Domain Model describes the data and the capabilities of a system. The Domain Model as well as the Task Model are concretized to an *Abstract User Interface Model* (AUI). The AUI is an abstract representation of the UI and comprises *Abstract Interaction Objects* (AIO), which are implementation and modality-independent representations of the UI elements. An example of an AIO would be the input element that receives some input from the user. In a graphical UI this AIO would be realized with an text entry field, in a speech based system it would be realized by a voice entry. Finally, the abstract representation in the AUI is concretized to a *Concrete User Interface Model* (CUI) that is a model-based representation of the model-specific implementation of the UI. From the CUI source code can directly be generated [Pleuss et al. 2010, 72].

[Pleuss et al. 2010] integrate the previously described concept of MBUID into the general SPLE processes *Domain Engineering* and *Application Engineering*. They propose to derive concrete GUIs of systems in Application Engineering on a higher level of abstraction, namely on the level of the the Task Model. Features of a product line are related to tasks in a Task Model. Thus, the AIOs are indirectly related to the features through their relation to the tasks in in the Task Model. At this point Pleuss et al. propose to give the software engineer the chance to customize the mapping from the AUI to the CUI and finally to the source code.

These customized models and mappings have to be individually tracked. Hence, the smaller the number of customized artifacts is, the smaller the effort to manage and track the customized artifacts is. A proposed technique the reduce the amount of artifacts that have to be customized is the ALM.

4.2 Auckland Layout Model

Today most GUI frameworks and toolkits contain layout engines that ease the positioning and configuration of widgets. Often they are feed with abstract information about the position and appearance of widgets and calculate the layout whenever it is necessary. Commonly used layout engines realize *row*-, *table*-, or *grid-bag* layouts. Configurations of these layouts usually cannot be reused and combined in a component-like fashion. Usually, this makes the configuration and synthesis of GUIs a hassle.

The ALM pursues another path. In ALM a GUI is described by means of constraints that have to be fulfilled by widgets. On the lowest level the GUI is modeled as an linear optimization program with constraints and an objective function that has to be minimized.

The constraints are formulated by means of tab stops between widgets of the GUI. Tab stops are either vertical (y-tabs) or horizontal (x-tabs). A layout with $x_0, x_1, \ldots, x_m \in \mathbb{N}$ x-tabs and $y_0, y_1, \ldots, y_n \in \mathbb{N}$ y-tabs would be formulated with constraints such as

$$a_0 x_0 + \ldots + a_m x_m + b_0 y_0 + \ldots + b_n y_n OP\ c$$

with the coefficients $a_0, a_m, b_0 \ldots, b_n$ and the right side c are real number and the operand OP is one of <, >, =. Constraints can be absolute, e.g. in terms of pixels, relative, i.e. in terms of space between widgets, or grouped to areas. These basic constraint patterns are used to build abstractions that are more powerful and more easy to use. Furthermore, it is possible to define other classes of constraints that take other parameters than the position of widgets into account [Lutteroth & Weber 2008, pp. 301].

The ALM is implemented for several GUI frameworks, such as Windows Forms. It is capable to separate the constraints from the actual application logic. It does this with an external configuration file that contains the constraints and a corresponding overloaded layout manager that wires the constraints with the actual application logic [Lutterot/Weber 2008, 301].

As the description indicates, specifying a GUI with constraints is more flexible than with common layout mechanisms. Furthermore, bears the division of the constraints from the actual GUI program code the potential to configure the GUI in a more component-oriented way. In how far these constrained based techniques can be used in generating GUIs for SPLs is topic of the proposed research and is planned to be tackled in the following way.

5 Research Outline

The planned research will investigate the above mentioned four research questions. Thus, the work will consist of four main work packages.

(1) Suitable constraint-based techniques have to be identified and selected.
(2) The selected techniques have to be adapted to the task of configuring and synthesizing GUIs in SPLs and possibly new ones have to be developed.
(3) A prototypical generator that facilitates the adapted and developed techniques has to be implemented.
(4) The developed approach has to be evaluated.

In the first work package suitable techniques will be identified through an extensive literature survey and evaluated according to a catalog of requirements for such techniques. In the second work package the identified techniques will be adapted and new ones will be developed. The underlying mathematical model will be based on the general structure of linear optimization programs with an objective function and several constraints.[Lutteroth et al. 2008] have already defined the general structure of linear programs to enforce usable GUIs in the ALM. However, since the requirements to enforce the usability for variable GUIs of products of an SPL differs from enforcing usability for single system GUIs, it is likely that some extensions are required.

In the third work package, a technique for constraint-based GUI generation for SPLs will be developed. The technique ought to be an extension of the work of [Pleuss et al. 2010] and should be integrated into the conceptual framework of GSD [Czarnecki/Eisenecker 2000] and its technology projection to EMP [Müller, 2009]. The general idea is to use the model based approach of [Pleuss et al. 2010] to describe the GUI. However, instead of directly generating GUI code, a GUI description, for example in ALM syntax, with annotated constraints is generated. Since the constraint-based definition of GUIs allows to express more complex relations between widgets of a GUI, it is possible that the GUI can adapt itself to the present widgets. This would not be possible with ordinary techniques such as grid-bag- or *table-based* layouts. To test the applicability of the developed methods and techniques a case study, for instance with the open source *Mobile Media* SPL [Figueiredo et al. 2008], will be carried out.

Within the case study the reuse infrastructure of the example will be adapted to the technology projection to EMP. That means, a domain specific language, a generator, and implementation components will be implemented or adapted. The generator will implement the adapted and developed techniques to configure and synthesize GUIs.

The prototypically implemented reuse infrastructure is the basis for the evaluation of the developed approach in the fourth work package. The following hypothesis shall be tested:

The adapted and developed constraint-based techniques generate usable GUIs for SPLs.

An empirical, quantitative evaluation is planned, which is carried out in three steps. First, a representative number of products is generated on the basis of the prototypically implemented SPL. Second, these products will be tested with common usability tests such as a cognitive walkthrough. It is intended that the tests are carried out by two groups of students trained in usability evaluation methods. One group will be from the University of Leipzig, Germany and the other group from the University of Auckland, New Zealand. Third, the results are interpreted and used to test the hypothesis.

6 Expected Contribution of the Work

With the planned artifacts, the research will be a contribution to industry as well as to science. Software manufacturers will get a first generator prototype that can serve as a basis for the development of commercial tools to automate the configuration and synthesis of GUIs in SPLs. Such a tool is able to increase the efficiency of software construction. Since many German software manufacturers – for instance *Intershop Communications*, *otris*, or *Delta Software Technology* to name a few – already employ SPLE techniques, they would become more competitive with such a tool on the global software market. From a scientific point of view, the work will show whether SPL development can benefit from constraint-based GUIs. The research will show, whether constraint-based techniques can ensure usable GUIs for SPLs.

7 Conclusion

In the paper we motivate the problem of automatic GUI synthesis and configuration in SPLE and present a literature survey on GUI generation for SPL and GUI generation approaches from the HCI community. We identified the work of [Pleuss et al. 2010]

where an approach to semi-automatically generate GUIs for SPLs is discussed. We further identified the ALM [Lutteroth et al. 2008] where an constraint-based approach to GUI specification is presented and discussed the potential to leverage the approach of [Pleuss et al. 2010] with the ALM.

The next step is to prototypically implement a generator to judge more sound on the usefulness of ALM for the GUI generation task in SPLE.

References

Abrams, M. et al., UIML: an appliance-independent XML user interface language. In: Comput. Netw. 31 (1999), pp. 1695–1708.

Apel, S. ; Kästner, C., An Overview of Feature-Oriented Software Development. In: JOT 8 (2009), Nr. 5, pp. 49–84.

Ardis, M. ; Daley, N.; Hoffman, D.; Siy, H. ,Weiss, D., Software product lines: a case study. In: Software-Pract. and Exper. 30 (2000), Nr. 7, pp. 825–847.

Balzert, H., Hofmann, F., Kruschinski, V., Niemann, Ch., The JANUS Application Development Environment - Generating More than the User Interface. In: Vanderdonckt, Jean (ed.): CADUI '96, Presses Universitaires de Namur, 1996. pp. 183–208.

Bergman, L.D., Banavar, G., Soroker, D., Sussman, J.B., Combining Handcrafting and Automatic Generation of User-Interfaces for Pervasive Devices. In: (Kolski u. Vanderdonckt, 2002), pp. 155–166.

Czarnecki, K., Overview of Generative Software Development. In: Banatre, J. P. et al. (eds.): Unconventional Programming Paradigms. Heidelberg, Berlin : Springer, 2005, pp. 326–341.

Czarnecki, K., Eisenecker, U.W., Generative Programming Methods, Tools, and Applications. Boston : Addison-Wesley, 2000.

Draheim, D., Lutteroth, Ch., Weber, G., Graphical user interfaces as documents. In: Proc. of CHINZ '06. New York, NY: ACM, 2006. pp. 67—74.

Figueiredo, E. et al., Evolving software product lines with aspects: an empirical study on design stability. In: Proc. of ICSE '08. New York, NY: ACM, 2008, pp. 261–270.

Kolski, Ch., Vanderdonckt, J. (eds.): Computer-Aided Design of User Interfaces III, 2002, Valenciennes, France. Kluwer, 2002.In: Constraints 13 (2008), pp. 307–342.

Lutteroth, Ch., Weber, G., Modular Specification of GUI Layout Using Constraints. In: Proceedings of the 19th Australian Conference on Software Engineering. Washington, DC, USA : IEEE Comp. Soc., 2008, pp. 300–309.

Müller, J., Generative Softwareentwicklung mit openArchitectureWare: Eine Fallstudie mit der E-Commerce-Plattform Intershop Enfinity Suite. Berlin: Logos, 2009 (Leipziger Beiträge zur Wirtschaftsinformatik 3).

Myers, B., Hudson, S.E., Pausch, R., Past, present, and future of user interface software tools. In: ACM TOCHI 7 (2000), pp. 3–28.

Oliveira, C., Rocha, F., Medeiros, R., Lima, R., Soares, S., Santos, F., Santos, I.H.F, Dynamic Interface for Multi-Physics Simulator. In: Inter. Journ. of Mod. and Sim. for the Pet. Ind. 2 (2008), Nr. 1, pp. 35–42.

Paternò, F., Santoro, C., One Model, Many Interfaces. In: (Kolski u. Vanderdonckt, 2002), pp. 143–154.

Pleuss, A., Botterweck, G., Dhungana, D., Integrating Automated Product Derivation and Individual User Interface Design. In: Benavides, David; Batory, Don; Grünbacher, Paul (eds.): Proc. of VaMoS ’10, 2010, pp. 69–77.