Towards More Reliable Configurators: A Re-engineering Perspective

Boucher, Quentin; Abbasi, Ebrahim Khalil; Hubaux, Arnaud; Perrouin, Gilles; Acher, Mathieu; Heymans, Patrick

Published in:
Proceedings of the 3rd Product LinE Approaches in Software Engineering (PLEASE'12), co-located with ICSE'12, Zurich, Switzeland

Publication date:
2012

Document Version
Early version, also known as pre-print

Towards More Reliable Configurators: A Re-engineering Perspective

Quentin Boucher, Ebrahim Khalil Abbasi, Arnaud Hubaux
Gilles Perrouin, Mathieu Acher
PReCISE, University of Namur
Belgium
{*qbo, eab, ahu, gpe, mac*}*@info.fundp.ac.be*

Patrick Heymans
PReCISE, University of Namur
INRIA Lille-Nord Europe,
Université Lille 1 – LIFL – CNRS , France
phe@info.fundp.ac.be

***Abstract*—Delivering configurable solutions, that is products tailored to the requirements of a particular customer, is a priority of most B2B and B2C markets. These markets now heavily rely on interactive configurators that help customers build complete and correct products. Reliability is thus a critical requirement for configurators. Yet, our experience in industry reveals that many configurators are developed in an ad hoc manner, raising correctness and maintenance issues. In this paper, we present a vision to re-engineering more reliable configurators and the challenges it poses. The first challenge is to reverse engineer from an existing configurator the variability information, including complex rules, and to consolidate it in a variability model, namely a feature model. The second challenge is to forward engineer a new configurator that uses the feature model to generate a customized graphical user interface and the underlying reasoning engine.**

***Keywords*-Configuration, Re-engineering, Graphical User Interface.**

I. Overview

Nowadays, both large and small companies adapt their production strategies to meet customization needs. To assist their customers in the customization, also termed configuration, of their products, they provide customization tools, more commonly called *configurators*. The cornerstone of advanced configurators is the reasoning engine that is responsible for keeping the configuration environment consistent by handling user's decisions, instantiating constraints, propagating the results and managing (usually preventing) conflictual decisions.

A growing share of these companies now propose web-based configurators, more than 800 of which are listed and categorized in 28 domains in [1]. The investigation of these configurators and our experience in industry reveal that existing configurators are developed in an ad hoc fashion: *(i)* the reasoning operations are not formally defined, raising correctness or runtime efficiency issues; *(ii)* the logical relationships between configuration options are hard coded, inducing severe maintenance overheads; *(iii)* the graphical user interface (GUI) itself is very rigid, hindering reuse across platforms (e.g., web-based vs standalone vs mobile) and its customization to user profiles.

Some of our partners expressed their need to migrate their legacy configurators towards more reliable, efficient, maintainable, and flexible solutions. This paper presents our vision to re-engineer web-based configurators (see Figure 1), and discusses the challenges we tackle. Our goal is to reach software as well as industrial (bikes, cars, etc.) domains.

The first challenge is to *extract configuration options and their dependencies*. Many heterogeneous artefacts are used during the reverse engineering process (Section II), e.g. GUI, web page source, code base, textual requirements, configuration files, etc. In this work, we focus on the elements accessible from the web page, i.e., the graphical widgets, the source of the web page, and its behaviour in order to extract variability information. We formally capture the output of the reverse engineering phase in a variability model. One of the most popular forms of variability model is the feature model (FM). Originally, FMs were developed in the context of Software Product Line (SPL) engineering [2] but our reengineering vision is agnostic regarding SPL methodologies. We chose the FM because its formal semantics makes it a very good candidate to pilot the configuration process and automate reasoning [3], [4]. Specifically, we rely on the *Text-based Variability Language (TVL)* to represent FMs [5].

The second challenge is to *produce a revised version* of the configurator. In this forward engineering phase (Section IV), the FM is augmented with styling directives to render the new GUI, and used to produce an API containing the reasoning engine (Section III). Together, the variability model, the styling properties, and the reasoning engine provide a generic, efficient and reliable solution. On top of these assets, designers can implement specific GUIs that respect the graphical standards imposed by a particular customer or platform. This latter stage is, however, not in the scope of this paper.

II. Reverse Engineering A Legacy Configurator

The migration of legacy applications is an age-old problem and different solutions have been proposed in several domains. To decrease the cost of migration and avoid creating new applications from scratch, most of these solutions are based on a reverse engineering process [6], [7]. We propose a semi-automatic and supervised reverse engineering approach to migrate a web configurator into a new configurator with an embedded constraint solver.

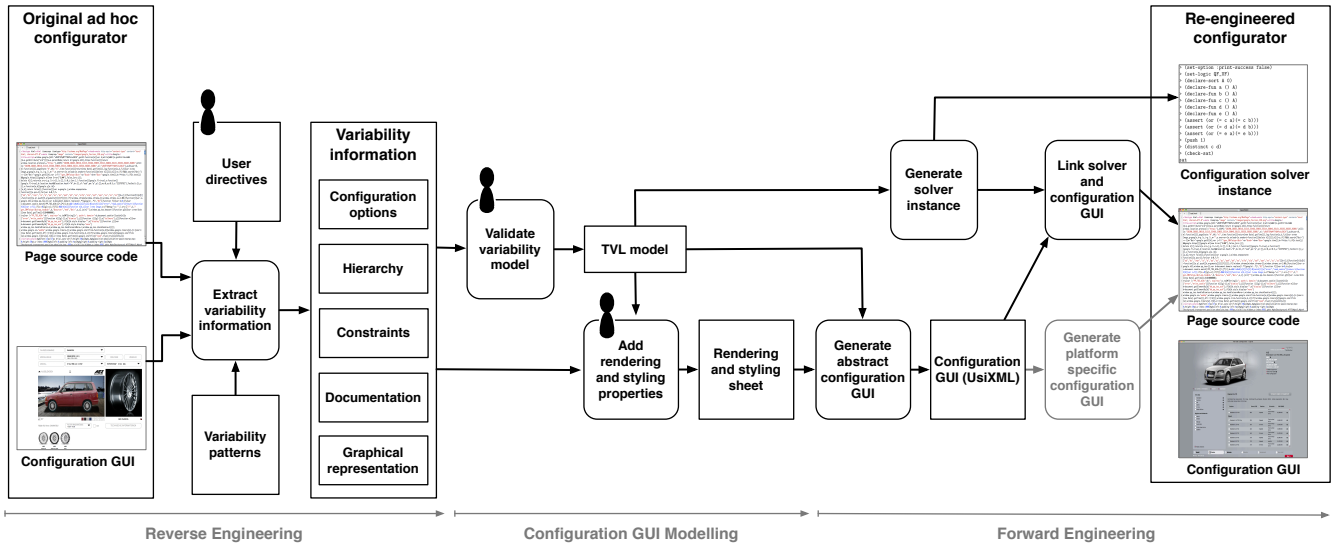

Figure 1. Re-engineering process

During reverse engineering, valuable configuration data (configuration options, hierarchy, constraints, etc.) is gathered by analyzing the existing GUI. As many yet different configuration GUIs exist, configuration data can be structured, organized and represented in a different way. Nevertheless *variability patterns*, i.e. elements of GUIs related to variability concerns that repeat in a predictable manner, can be identified, implemented, and reused within the reverse engineering process. A user can specify high-level directives to parameterize reverse engineering and make possible or improve the identification of variability patterns.

The goal of the reverse engineering process is to gather graphical widgets (e.g. HTML or jQuery elements and images) and determine their types (e.g. option, description field, and constraint). To guide the reverse engineering process, we sampled some existing web-based configurators from different industries and listed variability patterns that implement options and attributes, and mechanisms to describe and instantiate constraints. An example of variability patterns is using radio buttons to represent options contained in a group. It means that only one option can be selected in this group. This pattern is a directive to find options that are mutually exclusive in the configurator.

One important issue during reverse engineering is the detection of constraints. We discovered several constraint patterns. In one case, constraints are expressed in natural language and the element which contains this sentence is an inner child of the option element. In another case, constraints are wrapped into a jQuery object, which requires the execution of the associated code to actually observe the constraint it imposes. This task typically uses automated tools such as web crawlers.

Until now, our progress in supporting reverse engineering consisted in the design and development of a Firebug[1] extension. We chose Firebug because it is open source, and can be extended to support more functionalities. In essence the extension offers a search engine with ability to look for special patterns parametrized by the user. When a user browses a web page she can inspect which patterns are used and how they are implemented in the source code. Then, she initiates the search engine with these patterns and executes it. At the moment the search engine supports a few number of simple patterns and other patterns are currently being developed. To prevent the search engine from crawling the whole web page, the user can highlight a special part of the page and set the engine to just consider the highlighted region. The extension's settings also allow the user to define which attributes need to be recorded during the process. For example, the user might prepare the search engine to track attributes such as class, type, checked, etc. that usually hold valuable data about the GUI.

Reverse engineering yields an XML file that contains all extracted variability information. The extracted elements hierarchy is also kept unchanged. The XML model then is the input for the configuration GUI modelling part.

III. Configuration GUI Modelling

The textual elements extracted during reverse engineering are only option, attribute and constraint candidates. The user then has to validate this pre-typing, remove irrelevant elements, add new elements, rename elements, and recategorize elements. Currently, the post-processing step is supported by

[1] http://getfirebug.com/

an interactive GWT[2] application that lays out in a collapsible tree the updated model. Once the user is satisfied with the raw extracted data, a TVL model is generated.

A first step is to exploit the tree hierarchy to create and organize TVL features. This is an opportunity to refactor configuration options and to create new ones if needed. To identify the variability and constraints between these options, a set of patterns, derived from our survey, can be provided to the modeller as guidelines. For example, from constraint patterns, we know that a constraint can either be part of an option description (hint) or is its first inner child.

The next step is to add visualisation aspects to the TVL model. To cope with this issue, we encode this information in a property sheet linking GUI properties to constructs of the FM. This mapping is based on the retrieved GUI data in the pruned model. This property sheet is expressed in FCSS (Featured Cascading Styles Sheets), a CSS-like language we are currently designing. As in usual CSS, properties include layout information but also feature-specific visualisation strategies (e.g. hide, grey out, default value, etc.). Other properties may be related to the rendering of enumeration attributes depending on the number of available values, e.g. small ones may be represented as radio buttons in the final interface, while larger ones can be more efficiently represented as combo boxes. The availability of certain options may also depend on the target language (HTML, XUL, GWT, etc.).

We are currently designing an integrated textual editor for TVL and FCSS using the Xtext project[3]. In addition to syntax highlighting and checking facilities, Xtext eases the integration with the model transformation environment we exploit for the configuration interface generation.

IV. Forward Engineering A New Configurator

In this step, a set of model transformations needs to be applied to the integrated TVL-FCSS model to progressively derive the final configuration GUI in a given implementation language. So far, we have been experimenting model-to-text transformations (M2T) to generate configuration GUIs in the XUL language[4]. Based on MTL (MOF Model to Text Language), the upcoming OMG standard for M2T[5] and supported within the Acceleo environment[6], we defined a set of transformation patterns for mapping TVL constructs such as feature groups (or, xor, cardinality-based) and feature attributes (numbers, strings, enumerations...) to XUL widgets. These transformations patterns can be flexibly combined to generate the whole configuration GUI. Our initial experiments targeted Mozilla's XUL although these patterns are easily convertible to other XML-based user interface description languages. These patterns are platform-specific, though. We will evolve these patterns as model-to-model transformations between TVL-FCSS and UsiXML [8], a model-based standard for describing interfaces in a platform-independent way. Then, M2T transformations would be used to bridge the gap between the platform-independent model and the final GUI language. Additionally, we will devise solutions to incorporate existing designs (such as page templates) during the transformation process [9].

As noted in [9], integrating the configuration GUI with the reasoning code is a challenge. We offer a MVC-like architecture enabling the integration between the TVL model, the GUI and the configuration reasoning facilities. This explicit separation of concerns allows for more scalable configuration based on solvers in sharp contrast to ad hoc, hard-wired dependencies in most existing configurators. Furthermore, maintenance is greatly simplified by an independent variability model, FCSS and the automation of configurator generation.

V. Related Work

Reverse engineering the variability of existing systems has already been applied to different kinds of artifacts [10], [11], [12], [13], [14], [15], e.g., legacy system documentation, textual requirements, configuration files, source code, etc. For example, She *et al.* present procedures for reverse engineering FMs from operating systems [15]. Our work focuses on a specific kind of artifact: web-based configurators. We are not aware of existing works tackling the problem of reverse engineering variability models from such configurators.

Several tools provide support for reverse engineering GUIs but not for the specific purpose of retrieving variability. For example, Memon *et al.* [7] propose to reverse engineer web applications for the purpose of testing. Vanderdonckt *et al.* [16] developed *VAQUISTA* to reverse engineer a presentation model of a web page. In our work, the extraction of variability information from a web page is only the foundation stone upon which we build to re-engineer configurators.

Model-based generation of GUIs is an important research field in the human computer interface community. Several *Model-based User Interface Development Environments* (MBUIDEs) have been proposed. Each MBUIDE defines its own set of models to describe the interface. The different MBUIDEs and the associated models have been surveyed by Gomaa *et al.* [17] but none of them address the specific problem of generating configuration interfaces.

GUI generation based on FMs has been considered by existing works (see [9] for a more complete and detailed overview). For example, Pleuss *et al.* combine the concepts from MBUIDEs and FMs to automatically derive individual GUIs corresponding to selected features [18]. Schlee and Vanderdonckt [19] model the variability of the interface

[2]http://code.google.com/webtoolkit/
[3]http://www.eclipse.org/Xtext/
[4]https://developer.mozilla.org/en/XUL
[5]http://www.omg.org/spec/MOFM2T/1.0/
[6]http://www.eclipse.org/acceleo/

with an FM, which will be used to derive the corresponding GUI. Ultimately, our ambition is to include all the advanced feature of configurators such as beautification, configuration scheduling, views, etc. into the re-engineering process [9].

VI. Conclusion

The development and maintenance of configurators is a central yet difficult activity for many organisations. This difficulty is often amplified by the ad hoc nature of configurators in which the variability model, styling information and reasoning engine are all entangled. This paper presented our vision to re-engineering more reliable configurators and the challenges it poses.

Before implementing this vision, some other challenges still have to be dealt with. From a reverse engineering perspective, variability patterns should be identified in a more systematic way, formally defined and comprehensively implemented. The degree of automation and the quality of the models generated as output should be carefully evaluated. From a forward engineering perspective, different languages and transformations are still to be defined and combined. In particular, facilities to specify views (for structuring configuration screens) and feature configuration workflows [20] (for specifying configuration sequences) should be provided and integrated in the toolchain.

Our goal is to provide more systematic and flexible solutions for re-engineering legacy configurators. The huge number of existing configurators (e.g., see [1]) gives us a good opportunity to validate our procedures.

Acknowledgments

We thank our colleague Germain Saval for his suggestions and comments. This work was partially funded by the Walloon Region under the NAPLES project, the IAP Programme, Belgian State, Belgian Science Policy under the MoVES project, the BNB and the FNRS.

References

[1] http://www.configurator-database.com, 2011.

[2] P. C. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, ser. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[3] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-corts, "FAMA: Tooling a framework for the automated analysis of feature models," in *Proceeding VaMoS'07*, 2007, pp. 129–134.

[4] M. Mendonca, M. Branco, and D. Cowan, "S.P.L.O.T.: Software product lines online tools," in *Proceeding of OOPSLA'09*, 2009.

[5] A. Classen, Q. Boucher, and P. Heymans, "A text-based approach to feature modelling: Syntax and semantics of tvl," *Science of Computer Programming*, vol. 76, pp. 1130–1143, 2011.

[6] E. Stroulia, M. El-Ramly, P. Iglinski, and P. Sorenson, "User interface reverse engineering in support of interface migration to the web," *Automated Software Eng.*, vol. 10, pp. 271–301, 2003.

[7] A. M. Memon, I. Banerjee, and A. Nagarajan, "GUI ripping: Reverse engineering of graphical user interfaces for testing," in *Proceedings of RE'03*. IEEE, 2003, pp. 260–269.

[8] J. Vanderdonckt, "A MDA-compliant environment for developing user interfaces of information systems," in *Proceedings of CAiSE'05*. Springer, 2005, pp. 16–31.

[9] Q. Boucher, G. Perrouin, and P. Heymans, "Deriving configuration interfaces from feature models: A vision paper," in *Proceedings of VaMoS'12*. ACM, 2012, pp. 37–44.

[10] I. John, "Capturing product line information from legacy user documentation," in *Software Product Lines*. Springer, 2006, pp. 127–159.

[11] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler, "An exploratory study of information retrieval techniques in domain analysis," in *Proceedings of SPLC'08*. IEEE, 2008, pp. 67–76.

[12] N. Weston, R. Chitchyan, and A. Rashid, "A framework for constructing semantically composable feature models from natural language requirements," in *Proceedings of SPLC'09*, ser. ICPS, vol. 446. ACM, 2009, pp. 211–220.

[13] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire, "Reverse Engineering Architectural Feature Models," in *Proceedings of ECSA'11*. Springer, 2011, pp. 220–235.

[14] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions," in *Proceedings of VaMoS'12*. ACM, 2012, pp. 45–54.

[15] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "Reverse engineering feature models," in *Proceedings of ICSE'11*. ACM, 2011, pp. 461–470.

[16] J. Vanderdonckt, L. Bouillon, and N. Souchon, "Flexible reverse engineering of web pages with vaquista," in *Proceedings of WCRE'01*. IEEE, 2001, pp. 241–248.

[17] M. Gomaa, A. Salah, and S. Rahman, "Towards a better model based user interface development environment : A comprehensive survey," in *Proceedings of MICS'05*, 2005.

[18] A. Pleuss, G. Botterweck, and D. Dhungana, "Integrating automated product derivation and individual user interface design," in *Proceedings of VaMoS'10*. Universität Duisburg-Essen, 2010, pp. 69–76.

[19] M. Schlee and J. Vanderdonckt, "Generative programming of graphical user interfaces," in *Proceedings of AVI'04*. ACM, 2004, pp. 403–406.

[20] A. Hubaux, A. Classen, and P. Heymans, "Formal modelling of feature configuration workflows," in *Proceedings of SPLC'09*. Carnegie Mellon University, 2009, pp. 221–230.