EXTRA-FUNCTIONAL PROPERTIES FRAMEWORK WITH CONFIGURATION BASED ON DEPLOYMENT ENVIRONMENT

Tool Demonstration and Case-study

Kamil Ježek and Premek Brada
Department of Computer Science and Engineering, University of West Bohemia
Univerzitní 8, 30614 Pilsen, Czech Republic

Keywords: Extra-functional properties, Evaluation, Case-study, Tool, Component.

Abstract: Current software systems tend to share not only data but also software parts in a form of software components or web services. Once users cooperate third-party software parts, mechanisms and tools integrating and validating these software parts are desired. Therefore, advanced techniques for verifying compatibility and interchangeability should also consider extra-functional properties. This paper aims at supporting these techniques introducing a tool capable of definition, application and evaluation of extra-functional properties. The ability of the tool is demonstrated in a case-study developed on Common Component Modeling Example.

1 INTRODUCTION

Nowadays, industry and the research community deal with the issue of extra-functional properties (EFPs). For instance, (1) qualities such as speed, response time, memory consumption or (2) user requirements such as marketability, price, regular updates, technical support or (3) behaviour such as synchronisation, concurrent access, deadlock free computation are often addressed EFPs.

In our previous work an independent EFP framework (Ježek and Brada, 2011) enabling EFPs in existing industrial systems was proposed. This paper demonstrates that approach in a form of a case-study developed according to Common Component Modeling Example (CoCoME) (Herold et al., 2010).

This paper first introduces the case-study in Section 3 followed by domain EFP definitions in Section 3.1 and context-of-usage dependent EFP definitions in Section 3.2. Finally, Section 3.3 demonstrates the EFP evaluation process.

2 RELATED WORK

Other approaches target several disjunctive groups of EFPs. One group treats EFPs in isolation (Aagedal, 2001; Franch, 1998; Chung et al., 1999; ISO/IEC, 2001) while another consider their dependency (Zschafer and Meyerhfer, 2003). They provide basic ideas of a structure of EFPs, however, lack of tool support limits their practical usage.

There are structural frameworks with EFPs (Mohammad and Alagar, 2008) or systems with native support of either EFP or quality of service specifications (Yan and Piao, 2009; García et al., 2007) which still focus on a limited set of EFPs.

In addition, there are several component models (Becker et al., 2009; Muskens et al., 2005) which support specialised sets of EFPs. As in our work there are attempts to tool support (Sentilles et al., 2008) or Q-ImPrESS\(^2\) targeted at a selected set of component models.

In our previous work, we stated formal definitions of EFPs (Ježek et al., 2010) which extension and implementation we developed in (Ježek and Brada, 2011).

\(^2\)http://www.q-impress.eu/wordpress/
3 CASE-STUDY AND TOOL DEMONSTRATION

Case-study is in this paper based on CoCoME that is an example of a trading system with a set of hierarchical components. For the sake of brevity only few components have been selected to demonstrate the tool. The components composing the case-study are shown in Figure 1 with Notes to show attached EFPs.

3.1 Domain Dependent Definitions of Properties

The presented tool uses commonly accessible EFP repository spiting domain and context-of-usage concerns. For that, Global Registries (GRs) (Ježek et al., 2010) hold domain specific EFP definitions filled by domain experts.

Figure 2 shows a GUI editor of GR. The tool accesses a remote web-server storing the GR structure (Ježek and Brada, 2011). A user selects GR which he or she tends to work with, then EFPs are edited in terms of their names, data types, comparing functions and named values (in detail formalised in (Ježek et al., 2010)).

For our study, a GR named CoCoME with several EFPs (response time, express mode supported, bar code type, bar code scanner type) has been created. The EFPs defined in one GR bounds EFPs as valid and meaningful in one domain. Hence, the EFPs created for CoCoME cover the domain of trading systems.

3.2 Application of Property Values

Once EFPs are defined in Registries, the tool shown in Figure 4 is used for attaching selected values to components. Components are shown in one panel and
EFPs in the other one. A user drag-and-drops to attach EFPs.

Figure 4: EFP Assignment Tool.

The tool allows to assign three type of values (Ježek and Brada, 2011):

**Context Independent Values.** A lot of EFPs express values unchanged among all contexts-of-usage. Here, the tool assigns a concrete EFP value (e.g. number, string, boolean). An example of a property is the express mode supported attached to ExpressModeEnabledEvent. It is a boolean indicating the express mode of a cash desk which essentially has the same meaning all the time.

**Context Dependent Values.** Often values vary among contexts-of-usage. Here, several LR named values may be assigned. In the evaluation a user selects the LR to evaluator for. Such a property is the response time in the case-study. E.g. a cash desk may have harder performance needs than Inventory. Hence, the property values are defined in two separated LRs to distinguish two environments.

**Computed Values.** Complex properties are typically influenced each other (e.g. performance of one component is influenced by a connected components). Here, the tool adds mathematical or logical formulas computing an EFP value depending on other EFPs.

In CoCoME, the component Inventory::Database, Inventory::Data and Inventory::Application are used. Modeling EFPs, the property response time on the output of Inventory::Data is influenced by the output on Inventory::Database. A user may define e.g. a mathematical formula $\text{Inventory::Data: } \text{response time(PersistenceIf)} = 1.2 \times \text{response time(JDBC)}$

Another example uses the bar code type with values {EAN-8, EAN-13, UPC-A, UPC-E, ...} for existing bar codes. Bar code scanner type may be computed by a logical formula: $\text{bar code scanner type} = \text{EU} \iff \text{bar code _ type} = \{\text{EAN8, EAN13}\}$ where EU is the item of an enumeration.

### 3.3 Evaluation Process

When components are being assembled into an application, the EFPs of the provided and required features need to be evaluated for the compatibility. This is performed by an evaluator tool which, unlike other tools, does not have a GUI. It is rather implemented as a framework embeddable to other systems. It uses two steps to evaluate EFPs: (1) it binds components via their features and creates a graph expressing their connections, (2) the properties on bound features are compared by the algorithm from (Ježek and Brada, 2011).

For context independent values, the evaluator compares values directly attached on both connected properties. According to the case-study, if the CashDesk::CashDeskApplication requires the property express mode enabled set to true, then a compatible CashDesk::CardReaderController component has to provide express mode enabled also set to true.

Evaluation of context dependent values compares only values from a selected LR where a LR setting is a matter of configuration. Values irrelevant for other LRs are ignored. For instance, values on the properties response time for Inventory are taken only from Store LR while others are ignored.

Computed values evaluation uses depth-first search algorithm to recursively compute properties in a formula. In the use-case, evaluation on Inventory::Application::PersistenceIf needs to find Inventory::Database::JDBC first. Then a value of response time on Inventory::Data::PersistenceIf depending on Inventory::Data::JDBC may be computed. At the end, the values are compared as for direct values.

### 4 CONCLUSIONS

In this work a tool concerning extra-functional properties in terms of their definition, application and evaluation has been presented. The tool is based on a framework, developed in our previous work.
The tool aims at improving adoption of extra-functional properties in practice. This aim has been demonstrated by the presented case-study which comprises a part of the Common Component Modeling Example proving the framework’s applicability to a practical application.

The case-study has shown that the system of registries is capable of managing extra-functional properties on systems deployed in different runtime environments. On the other hand a few weakness have been found. Namely, there are properties which need more sophisticated evaluating methods (for instance, memory consumption is not a simple summation of each component if they use shared libraries). Such evaluating methods we will target in the future. In addition, filling of values in LRs is a time consuming manual work. For that reason formulas transferring values among LRs would improve the approach.

ACKNOWLEDGEMENTS

We would like to thank students Martin Štulc, Lukáš Vlček and Jan Sváb for their considerable help in preparing the presented case-study.

REFERENCES


