Usability Evaluation of Configuration-Based API Design Concepts

Thomas Scheller and Eva Kühn

Institute of Computer Languages
Vienna University of Technology
1040 Wien, Austria
{ts,eva}@complang.tuwien.ac.at

Abstract. Usability is an important quality attribute for designing APIs, but usability-related decision factors are often unknown. This is also the case when looking at APIs for configuration tasks, like for dependency injection or object-relational mapping. In these areas three different API design concepts can be found, which are annotations, fluent interfaces, and XML. There exists no research concerning usability-related characteristics and differences between these concepts.

In this paper, we present a usability study that identifies such characteristics and differences between the three concepts, by comparing three different variants of an API for dependency injection. From the study results we evaluate advantages and disadvantages in different use cases, and show how to build more usable configuration-based APIs.

Keywords: API Usability, API Design, Fluent Interfaces, Annotations, XML.

1 Introduction

There are many areas in programming where APIs (application programming interfaces) are used for some kind of configuration. Examples are APIs for object-relational mapping like Hibernate[1], APIs for dependency injection like Unity[2], or APIs for communication like the Windows Communication Foundation[3] (see Table 1). Next to logging and unit testing, these are the most prominent areas where external APIs are used. When looking at such "APIs for configuration", many of them share the same basic design concepts, independent from the area of usage. We identified three design concepts that are used for such APIs: The first is *XML*, where the whole configuration is not written in code, but stored in a separate XML file. The second is *annotations*, where the configuration is done by annotating code elements with additional information. The third is *fluent interface* [1], which has only recently become popular, and tries to make use of the natural language by defining methods that are concatenated to form a readable sentence.

[1] http://www.hibernate.org
[2] http://unity.codeplex.com
[3] http://msdn.microsoft.com/en-us/library/dd456779.aspx

A. Holzinger et al. (Eds.): SouthCHI 2013, LNCS 7946, pp. 54–73, 2013.

Table 1. Examples for configuration-based APIs

API for	Configures	Then does
Dependency Injection	bindings and injections	create instances
Serialization	which fields are serialized	serialize/deserialize objects
Object-Relational Mapping	mappings from code to db	insert, update and read data to/from the db
Communication	which messages are sent, which transport layer is used	send/receive messages

When evaluating the usability of such an API, it is therefore important to understand the usability implications of the used design concept(s). There is not yet any research concerning the usability of XMLs, annotations, or fluent interfaces in this context. Some existing papers deal with the usability evaluation of APIs [2,3,4,5]. They analyze the impact of parameters in objects' constructors, compare the usability of constructors and static factory methods, check the implications of method placement on API learnability, and evaluate the usability implications when using classes and methods in different scenarios. These papers cover well the usage of basic elements of object oriented APIs, but do not take higher level design concepts into account like the ones mentioned above. In the context of XML there are a few papers dealing with usability [6,7], but they are either too problem specific or not applicable in the context of programming. Further, there is a number of API design guidelines like [8] and [9], which give a good overview how to build usable APIs. While such guidelines show how to build APIs with each of the mentioned design concepts, there are none comparing them and/or saying when it is best to use which one. Further, they don't have a scientific basis. Today, when a programmer e.g. wants to choose a dependency injection framework, he/she has a large selection of different APIs with different design concepts. It has often been shown that usability is an important factor [10,11,12], both for API developers who need to know how to design an API, as well as for API users who want to choose the best usable API. But usability can only be taken into account if the according differences between design concepts are known.

Therefore, in this paper we want to evaluate the usability of the three API design concepts *XML*, *annotations* and *fluent interface*, and compare them to each other. We therefore conducted a usability study with three different APIs, each implementing one of these design concepts. As a result of the study, we want to show which designs perform better or worse in which situations, and which factors influence usability for each design concept.

In section 2 we present the design of the study, including the used APIs and details about the test execution and data analysis. Section 3 presents the study results, with statistical evaluations for each measured performance detail. An interpretation of the results is shown in section 4, where we present advantages, disadvantages and measurable properties for each design concept.