HAL
open science

Testing Executable Themes

Andrew Jackson, Jacques Klein, Benoit Baudry, Siobhan Clarke

To cite this version:

Andrew Jackson, Jacques Klein, Benoit Baudry, Siobhan Clarke. Testing Executable Themes. In Second Workshop on Models and Aspects, Handling Crosscutting Concerns in MDSD at ECOOP 06, 2006, Nantes, France, France. inria-00512543

HAL Id: inria-00512543

https://inria.hal.science/inria-00512543v1

Submitted on 30 Aug 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Testing Executable Themes

Andrew Jackson[1], Jacques Klein[2], Benoit Baudry[2], Siobhán Clarke[1]

[1]Distributed Systems Group, Dept. of Computer Science, Trinity College Dublin, Ireland.
{Andrew.Jackson, Siobhan.Clarke} @cs.tcd.ie
[2]IRISA, Campus Beaulieu, 35042 Rennes Cedex, France.
{jklein, benoit.baudry} @irisa.fr

ABSTRACT

Design validation is important for detecting errors early in the development life cycle. Testing the design is one significant means to achieve design validation. In this paper we present initial work to make Theme/UML models executable and therefore testable.

1. INTRODUCTION

Design validation is important for detecting errors early in the development life cycle. The earlier an error is detected, the easier and cheaper it is to resolve. Testing the design is one significant means to achieve design validation [3].

There are various ways to design a system. One popular means to mitigate errors is to employ test driven development. Typically tests are formulated based on use cases which often represent concerns. The concern decomposition facilitated by use case modelling is generally lost when using traditional design paradigms (e.g. object-orientation). This is because these paradigms do support concern modules as a first class construct in design.

When concerns are scattered and tangled in one monolithic design, the design becomes harder to test because it is harder to write a test case that targets one concern in complete isolation. If concerns are scattered and tangled an error in the design of one concern can have a negative impact on other concerns with which it is entangled. As such, it is difficult to detect the error and localize the effect of resolution.

The UML enables the designer to separate some kinds of concerns. Aspect oriented modelling (AOM) approaches typically extend the UML increasing the scope for concern separation at design time. It is claimed that concern separation improves design reusability, compensability and flexibility [2]. We are investigating the extent to which concern separation at the design level also improves the testability of design. Design models that represent concerns are focused on one area of interest in a system. Through this work we are investigating whether it is easier to express scenarios as test cases for a specific concern. Moreover, we expect that when an error is detected; it will be easier to identify the precise causes of errors and resolve them quickly.

Various AOM approaches exist for separating concerns within the design space [4]. Theme/UML [1] is unique within this space as it provides both a symmetric decomposition model and well defined composition semantics. A symmetric decomposition model ensures that both crosscutting and non-crosscutting concerns can be modularized. In Theme/UML crosscutting and non-crosscutting concerns are modularized as themes. Themes encapsulate the standard UML structural and behavioural diagrams required to capture the concerns structure and behaviour. Well defined composition semantics describe the effect the composition operator (e.g., merge) has on themes.

Testing is checking the consistency between what the developer wants and what the designer has. Typically this would be the execution of a test case against a program. In our case, we need to be capable of comparing two views on the same design model (what the developer wants and what the designer has). Theme/UML provides two views that would allow us to test a Theme model (behavioural and structural diagrams). Behavioural diagrams have been illustrated as a good means for generating test cases [3]. They define some particular expected traces through a system based on a defined context. The behavioural diagrams for a particular theme can be used to describe test cases. For a theme to be testable, the structural diagrams need to be executable.

In our initial work we are augmenting the Theme/UML model to be executable. We have implemented a prototype Theme/UML model on the Kermeta platform[1]. The Kermeta platform enables the weaving of executability into object-oriented meta-languages. Theme/UML extends the UML by defining the composition of UML models, modularized in themes. By implementing the Theme/UML model in Kermeta we are able to weave executability into themes and define the composition of executable themes.

Kermeta enables executable behaviour to be woven into structural models [6]. In Theme/UML behaviour is typically described through behavioural diagrams. When themes are executable, diagrammatic representation of behaviour is no longer necessary as the behaviour may be observed through model execution. In existing work, the weaving of scenario models has been investigated and formally specified [5]. Scenario composition is also being implemented on the Kermeta platform.

In this paper we propose the modularization of test scenarios and executable theme models within a new KerTheme model. Like the original Theme/UML model the KerTheme model provides two perspectives the executable theme model describes what the designer has and the accompanying scenarios describe what the designer wants.

It is our intuition that we will be able to test theme designs by making themes executable and defining test scenarios for these executable themes. By comparing scenarios and execution paths through theme models it may be possible to validate a theme in isolation. However, in most cases themes need to be composed to ensure correct execution. In this paper we outline a means for a consistent merge of executable class diagrams and sequence diagrams to generate both a composite test case and composite theme model, whereby the composite test case fully validates the composite theme model.

2. MERGING MODELS AND TESTS

In this section we present a simple case to illustrate our position. This case is based on an Auction System that has been used in previous works e.g. [4].

---

[1] http://www.kermeta.org/

Figure 1 illustrates two executable themes LogIn and Persistency. The «KerTheme» stereotype denotes themes as executable and testable. Each KerTheme contains both executable classes (executability is represented by a lightening symbol) and sequence diagrams which represent test case scenarios. An executable class is one where by execution logic (expressed in the Kermeta Language) is woven into the methods declared on the meta-class.

The LogIn KerTheme is testable because the logic encapsulated in the executable classes models and the resulting execution path and result can be compared against the sequence diagram test cases. This is not the case for the Persistency KerTheme, as both KerTheme class model and sequence diagram test cases are parameterized and incomplete until bound through composition.

To test the persistency behaviour we need to merge both the executable class model and the sequence diagrams test cases with the LogIn executable class model and sequence diagrams test cases. Figure 2 illustrates the composition of the LogIn and Persistency KerThemes. In this diagram the executable class models are unified and the sequence diagrams are composed. Through comparing the execution path of this composed model against the sequence diagram test case we can ensure that the logic that is described in the persistency KerTheme is error free. We can also regression test the composite Auction KerTheme with the sequence diagram test case defined in LogIn KerTheme.

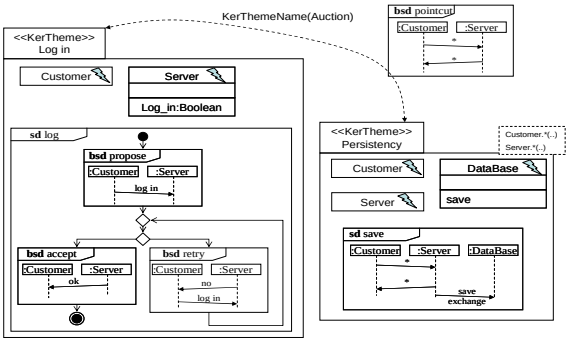

Figure 1 Merge (Please see [1] and [5] for precise description of notation)

To realize this composition two separate merge operators have been developed. One merge operator has been developed to compose executable class models this and is based on the unification of semantics defined for Theme/UML [1] and PackageMerge in UML 2.0 [7]. The other has been developed to merge sequence diagram test cases. The semantics of merging sequence diagrams are defined in [5]. The semantics for merging now needs to be synchronized in such a way that both models and test cases can be consistently composed.

To achieve theme testability we also need some means for capturing execution paths through models and comparing these with sequence diagram test cases. We have begun an investigation into how this may be achieved within the model that we have proposed implemented on the Kermeta platform.

3. CONCLUSIONS & FUTURE WORK

Although work is being done in the area of testing in AOSD [8], little of this work is focusing on the model level. An innovation of our approach is to take a well defined symmetric AOM model and enhance this model with means to test individual concerns through the definition of simple tests. Another innovation of this work is the definition of a merge operator that synchronizes the merging of both models and the tests with which correspond to these tests. By composing tests we can execute tests and models and test composite tests against composite models and evaluate the result. Where the results are negative there may be problems with the composition as specified.

In our future work we will further investigate and extend the proposition presented in this paper. We will continue to investigate other means for testing AO based models at design time. We will also evaluate our approach through the Auction Case study described and used in [3] and [4].

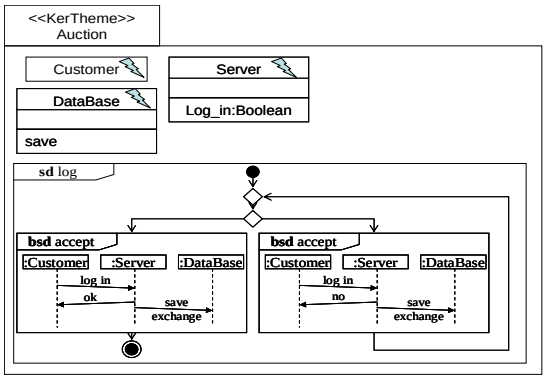

Figure 2 Composite

ACKNOWLEDGMENTS

This work is supported by European Commission grant IST-2-004349: European Network of Excellence on Aspect-Oriented Software Development (AOSD-Europe), 2004-2008

REFERENCES

[1] Clarke S., Baniassad E., "Aspect-Oriented Analysis and Design the Theme Approach", ISBN: 0321246748, 2005 Addison-Wesley

[2] Clarke S., Walker R. J., "Composition Patterns: An Approach to Designing Reusable Aspects", 23rd International Conference on Software Engineering (ICSE), Toronto, Canada, May 2001.

[3] Dinh-Trong T., Kawane N., Ghosh S., France R., Andrews A. A., "A Tool-Supported Approach to Testing UML Design Models", 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), June 2005, Shanghai, China

[4] Jackson A., Clarke S., "Initial Version of Aspect-Oriented Design Approach, aosd-europe.net, Feburary, 2006

[5] Klein J., Helouet L., Jézéquel J.M., "Semantic-based Weaving of Scenarios". 5th International Conference on Aspect-Oriented Software Development (AOSD' 06), March 2006. Bonn, Germany

[6] Muller P.A., Fleurey F., Jézéquel J.M., "Weaving executability into object-oriented meta-languages", Proceedings of MODELS/UML'2005, volume 3713 of *LNCS*, pages 264-278, October 2005, Montego Bay, Jamaica

[7] UML Superspec p107-115, http://www.omg.org/, 2004.

[8] Xu D., Xu W., "State-Based Incremental Testing of Aspect-Oriented Programs", 5th International Conference on Aspect-Oriented Software Development (AOSD' 06), March 2006. Bonn, Germany