KerTheme: Testing Aspect Oriented Models

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

[1] Distributed Systems Group, Computer Science Dept., Trinity College Dublin, Dublin, Ireland.
{andrew.jackson, siobhan.clarke}@cs.tcd.ie
[2] IRISA, Campus Beaulieu, 35042 Rennes Cedex, Rennes, France.
{jacques.klein, 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 introduce the KerTheme model. KerTheme provides a means for symmetrically decomposing concern based executable class diagrams and concern test scenarios. KerTheme also facilitates synchronised merging of these decomposed models into a coherent composite concern based executable class model and corresponding test scenarios. The KerTheme model allows us to investigate whether decomposed concern based executable class diagrams simplifies the definition of concern test scenarios. This will also allow us to investigate whether this approach ensures more rigorous testing of a complete system.

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 [6]. When concerns are scattered and tangled in one monolithic design, the design becomes harder to test. This is 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 [5]. Various AOM approaches exist for separating concerns within the design space [7]. Theme/UML [4] is unique within this space, as it provides both a model which supports symmetric decomposition 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 can be seen as an activity that consists of 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 [6]. 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.

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 if it is easier to express scenarios as test cases for a specific concern. We are also investigating whether it is possible to synchronize tests and concern composition such that tests and their associated concerns can be composed into full and coherent system models which are fully covered by composed tests. Moreover, we expect that when errors are detected, it will be easier to identify the precise causes of these errors and resolve them quickly.

To facilitate this investigation, we introduce the KerTheme model. The KerTheme model supports the decomposition of concerns and concern tests into KerThemes. A KerTheme comprises of an executable class diagram and test scenarios. KerThemes are units of composition. The composition semantics for executable class diagrams are based on composition semantics defined in Theme/UML. The scenario weaving approach [8] is used to compose test scenarios. To ensure that both executable class diagrams and test scenarios are merged in a synchronized manner, we are aligning the merges defined for scenarios and executable themes.

In this paper we motivate and illustrate the Theme/UML, Scenario Weaving and KerTheme models through an Auction System Case study[1]. This case study has previously been used in [7, 8]. In this paper we focus on two concerns, a login concern and a persistence concern, which are part of the Auction System. The persistence concern crosscuts the login concern at points where login attempts are made. The persistence concern deals with the recording of login attempts.

In section 2 we describe composable executable class diagrams for building testable models. Section 3 describes why scenarios are good for defining tests and how scenarios can be composed. Section 4 introduces the KerTheme model and describes how this model both supports concern testing and synchronised composition of concern models and tests. Section 5 outlines the benefits and limitations of this approach and also includes a brief discussion. Finally, Section 6 concludes the paper and outlines future work.

[1] A description of the Auction System is available at http://lgl.epfl.ch/research/fondue/case-studies/auction/problem-description.html

2 Executable Class Diagrams

Theme/UML [4] is a MOF based extension [5] to the UML that supports AOM. Theme/UML facilitates the symmetric concern based decomposition of a system and the specification of how (base and aspect) design modules are to be composed. In Theme/UML composition is specified as a merge relationship between themes. The top half of Figure 1 illustrates the design of the login and persistence concerns as themes. The login theme is a *base theme* and the persistence is an *aspect theme*. A *base theme* is an extension of the UML package meta-class, instances of which encapsulate the structural and behavioural UML diagrams that the designer needs to describe a concern. An *aspect theme* extends the template package meta-class. The *template parameters* associated with the template package are used in the description of *crosscutting behaviour*. The *template parameters* are representative of any join points affected by *aspectual behaviour*. In the login and persistence themes at the top of Figure 1, class and sequence diagrams are used to describe the structure and behaviour of concerns.

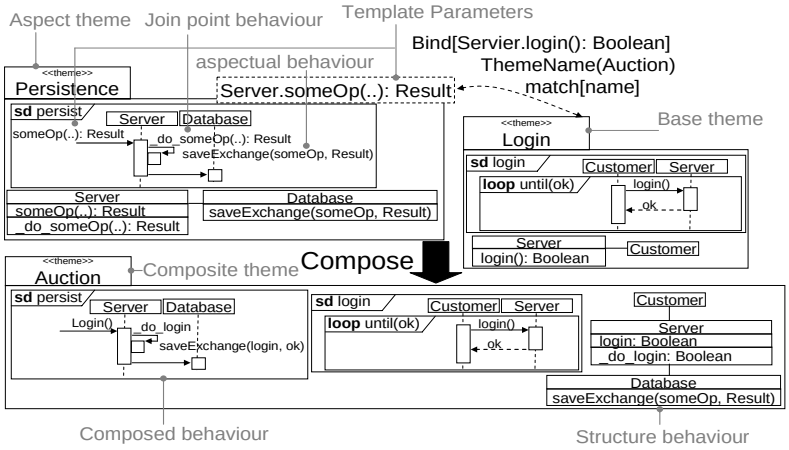

Figure 1 Composition of login and persistence themes

Figure 1 also illustrates the specification of a *merge* composition of the persistence and login concerns. As persistence is crosscutting the merge specification a *join point binding* is specified as part of the *merge*. A join point binding specifies elements that exist within themes as points that are to be crosscut. In Figure 1 the `Server.login(): Boolean` method is bound to the template parameter exposed by the persistence theme. The result if this binding is presented at the bottom of Figure 1 in the Auction *composite theme*. The class diagrams that were defined in the persistence and login themes are unified into one class diagram. There are two sequence diagrams in the composite theme. The login sequence diagram (SD) is equivalent to the login SD in the login theme. The persist SD shows how the join point specified in the *join point binding* replaces the *template parameter* to describe composed behaviour.

KerMeta [9] is a meta-modelling language that has been designed as an extension to the EMOF 2.0 to be the core of a meta-modelling platform. KerMeta extends EMOF with an action language[2] that allows the specification of behavioural semantics formetamodels. This action language is imperative and object-oriented. It is used to provide an implementation of operations defined in meta-models. As a result the KerMeta language can, not only be used for the definition of meta-models but also for implementing their semantics, constraints and transformations.

KerMeta has been used in the successful implementation of class diagram composition in [15] but also as a model transformation language in [10]. Executable Theme/UML and more specifically executable class diagrams have been defined in a manner similar to [15]. Executable class diagrams are constructed by defining structural meta-models (similar to class diagrams in Theme/UML) wherein the themes behaviours (defined in behavioural diagrams, such as sequence diagrams in Theme/UML) are defined in the methods of the structural meta-models with the KerMeta action language. The merge semantics for executable class diagrams are defined within a merge composition operator which defines the composition of both structures and behaviours modelled in KerMeta. The merge has been architected as prescribed in [7]. When the merge operator is applied to executable themes, the result is a composite executable theme model in which the structural meta-models are composed and the behaviours modelled in the KerMeta action language are composed (similar to composite themes).

3 Weaveable Scenarios for Concern Test Cases

A number of works study the use of sequence diagrams (which represent scenarios) to define and generate test cases. In [6], the authors propose a technique to automatically generate test cases for UML design models. The UML design models consist of a class diagram, OCL pre and post conditions for methods and activity diagrams that model the behaviour of each method. From this design model, the authors can generate an executable form of the model, which can be tested using dynamic testing techniques. Concerning test generation, they propose to model test cases using UML2.0 sequence diagrams. From these test cases specification and the class diagram, they generate a graph that corresponds to all possible execution paths defined in the different scenarios. The authors then use coverage criteria, defined in [1]. From the graph, it is possible to automatically generate test data and an initial system configuration to cover each execution path.

Other works also propose to use scenarios as a basis to generate cases for programs. In [11, 12], Pickin et al. investigate the use of sequence diagrams as a formal language to write test cases for distributed systems. The UML model of the system has to be composed of a class diagram and a state diagram. Then, from test objectives, modelled with sequence diagrams and a description of the initial state of the application in the form of an object diagram, the authors propose a technique to synthesize test cases for the system. The approach uses input/output labelled transition

[2] A complete description of the action language definition can be found in [9]

systems (IOLTS) as an intermediate formal model from test cases are generated. Thus, they have a transformation from a UML model to IOLTS, and a reverse transformation to represent the generated test cases with sequence diagrams. The test cases specify the ordering of call sequences that should be sent to the system and associated test verdicts.

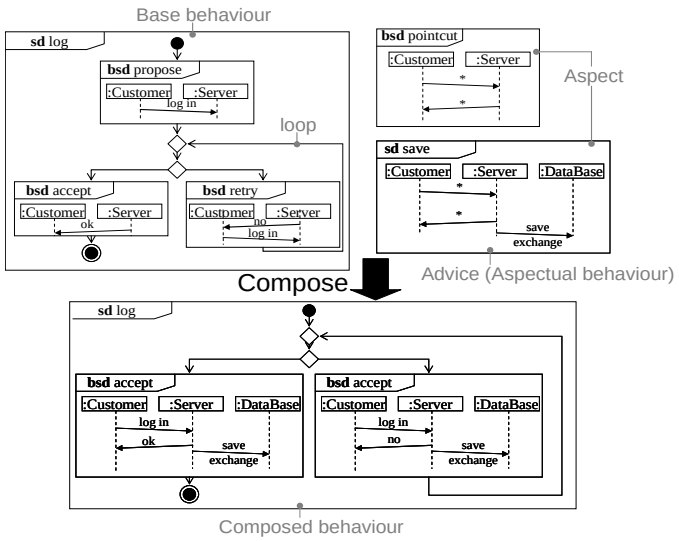

Figure 2 Scenario Weaving

The Object Management Group also considers sequence diagrams as a practical language to write test cases for object-oriented systems. The UML testing profile [14] proposes a meta-model that captures all the concepts needed to design and generate black-box test cases: test objective, test case, test data, test environment, system under test (SUT)... In this profile, the behaviour of a test case (the behaviour a test case aims at validating) is described using a sequence diagram. The profile also includes mappings from test scenarios to JUnit and TTCN.

In [2], Briand et al. propose to use scenarios associated with use cases as high level test cases. The authors propose to use activity diagrams to model the ordering of use cases. It is then possible to extract paths from the activity diagram, which correspond to legal sequences of use cases. Once these sequences are available, each use case is replaced by its corresponding sequence diagram to build a global test case. The generated test cases correspond to requirements level test cases that should be executed on the final implementation of the system.

In [8], Klein et al. propose a semantic-based weaving of scenarios, where the weaving is based on the dynamic semantics of the models used. An example of this approach is presented in Figure 2. This example illustrates the modelling and composition of the login and persistence scenarios[3]. In this approach, an *aspect* is described

[3] Please note that the scenario models and composition specification is similar to the Theme/UML model.

as behavioural aspect because it is specified with behavioural modelling languages. An aspect, in this approach, is defined as a pair of SDs, one SD for the *pointcut* (specification of the behaviour to detect), and the second one for an *advice* representing the expected behaviour at the join point. Similarly to Aspect-J[4], where an *aspectual behaviour* can be inserted 'around', 'before' or 'after' a join point, with the approach defines in [8], an *advice* may indifferently complete the matched behaviour, replace it with a new behaviour, or remove it entirely to create *composed behaviour*.

One of the difficulties in weaving SDs is that dynamic behaviour needs to be woven at modelling time. Therefore, we need to statically find where the join points are in the *base behaviour*. While this can be trivially implemented with a syntactic match for simple SDs, the hierarchical nature of UML 2.0 SD (similar to HMSCs [16]) makes it necessary to address the problem at the semantic level [8] with static analysis techniques such as *loop* unrolling, etc… In the next section, the weaver proposed in [8] is used to compose scenarios (represented as sequence diagrams).

4 Merging Models and Tests: KerThemes

To facilitate our investigation into the extent to which concern separation at the design level improves the testability of design, we are integrating executable class diagrams (described in Section 2) and weaveable scenarios (described in Section 3). As mentioned in Section 1, testing is checking the consistency between what the developer wants and what the designer has. It is expected that symmetric concern decompositions will make it easier for the developer to describe what he/she wants in from a specific concern. Once developed, an executable class diagram is what the designer has to test. For defining what the designer wants (or the test), we use scenarios. The executable theme model is thus really executable since both the global structure and the behaviour are defined. Moreover, a KerTheme contains a set of scenarios which correspond to behaviours the designer would like to see. Our claim in this paper is that these scenarios can serve as a basis for test case generation to test the executable class diagram. A KerTheme is thus said to be testable. This is because it contains an executable design model of the theme and scenarios that can be used to validate this theme.

The KerTheme model also facilitates the *synchronised merge* of both concern model and concern test scenarios. This is achieved by ensuring that the semantics of behavioural composition in Executable Theme/UML are consistent with those defined for Weaveable Scenarios.

Figure 3 illustrates an example of the KerTheme model. In this model, there are two KerThemes: LogIn and Persistence. Concern models denoted with the «KerTheme» stereotype are defined as being executable and testable. Each KerTheme contains both executable class diagram (executability of classes is represented by a lightening symbol) and sequence diagrams, which represent test case scenarios.

[4] http://www.eclipse.org/aspectj/

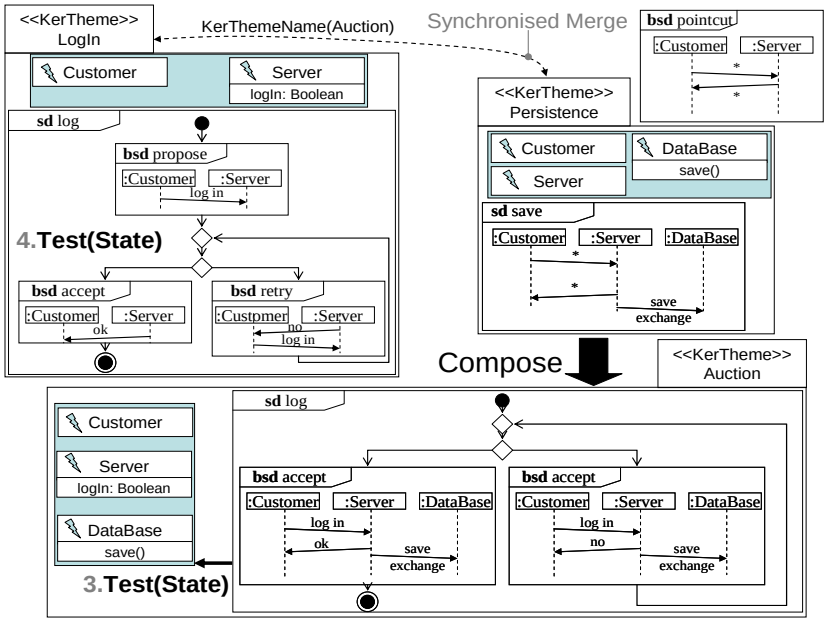

Figure 3 KerTheme models and composition

KerTheme provides an effective approach for system modelling and design, thanks to a clear separation of concerns. It also offers an interesting solution for test case generation. Using this method, it is possible to model test cases for each concern separately without, caring about the complex interactions that might appear in the global design. For example, in Figure 3, it is possible to describe the expected behaviour of the Persistence concern with a very simple model. Each time messages are exchanged between the client and the server, this exchange should be saved. When describing this abstract test case, we do not need to consider every case where such an exchange can occur; we really focus on the expected behaviour of the single concern.

However, since each concern is modelled separately, these models have to be merged to get a model of the global behaviour of the system. This, in turn, means that the executable class diagrams have to be merged to get a global design and that the scenarios also need to be merged to obtain consistent test cases for the system. Indeed, in most cases, it is not possible to run the test case directly on the concern model, independently from the rest of the model.

The LogIn KerTheme is an example where it may be possible[5] to test the theme independently. It is possible because an initialization step may be modelled in the executable class diagram for bootstrapping the auction system elements. Using the sequence diagram associated with this concern, we can generate at least two test cases: one with positive test case and negative test. We could then run those test cases

[5] Customer and server objects may need to be initialised.

against the executable class diagram and check that it behaves correctly in these cases.

The Persistence KerTheme can not be solely tested using the information given in the model. Since this is a cross-cutting concern, both KerTheme class model and sequence diagram test cases are parameterized and incomplete until bound through composition. This facilitates the development of a simplified, generic test case that could take into account any exchange between client and server that would appear at any time during execution. To be able to execute the class diagram and run the test case, we have to merge both the executable class model and the sequence diagrams test cases with the LogIn executable class model and sequence diagrams test cases. The bottom of Figure 3 illustrates the composition of the LogIn and Persistence KerThemes. In this diagram the executable class models and the sequence diagrams are composed in a synchronized manner by using the weaving process presented in section 2 and 3. It is then possible to generate at least two cases, one positive test, where a login is performed correctly and one negative test, where a bad login in performed. In both cases, a check to ensure that the exchange is saved is required. Then, when running these test cases against the design model, we can validate that the executable classes actually conform to the expected behaviour.

We can also regression test the composite KerTheme with the sequence diagram test cases defined in LogIn KerTheme. When test scenarios are considered to define contracts for an executable class model, then the result from the application of both woven and unwoven test scenarios to a composite executable class model should be consistent.

5 Discussion

We believe that this approach shows merit. More work is required to investigate how comparisons of scenarios and execution traces through a system may be realised. We have identified several benefits and limitations of this approach.
The benefits include:

1. **The correctness of the design model can be validated**: the designer can be more confident that errors in the design logic will not emerge during Aspect and Object oriented implementation.
2. **Test cases are easier to create and change**: the designer can focus on creating tests for one concern in isolation and as such tests are more concentrated and easier to change.
3. **Finding errors is easier**: when a test fails it is possible to relate the error to a particular concern. As we check execution paths against sequence diagram test cases, it should be easier to identify where errors arise.
4. **The affect of fixing errors in minimized**: Changes to the design model are localised within a KerTheme and as such the negative effects of change may be reduced.
5. **Improved Reusability**: As KerTheme represents a modular unit of design logic and tests for that design logic. As such, KerThemes are easier to reuse.

The limitations we have identified include:

1. **May need global tests**: Untested behaviour may emegre in composite models. This may be due to interactions between inputs that form the composite. It may be difficult to write test cases in isolation to test these behaviours. In some cases it is necessary to add global test cases for the global design to ensure tests cover these types of behaviours.
2. **Weaving correctness**: It may be difficult to ensure that tests and models have been composed correctly, unless additional tests are applied to the composite. For this approach to succeed, we need to validate the composition tools. Such that, if an error is detected in the global composite design, we know it comes from one of the models and that it is not an error introduced by the weaving mechanisms.
3. **Join point selection**: It may be difficult to assess whether a join point designator is correct. This is because test flows through composite tests may lead the execution flow away from the operations where join points may emerge.

Another issue that is of interest is the difference between an executable class diagram and an object-oriented program. An executable class diagram is not the same than an object-oriented program. Firstly, with an executable class diagram, the level of abstraction is higher: it is independent of a platform. Secondly, when a class diagram is implemented with an OO language, the initial model is not really preserved. For instance, the associations between classes are changed into attributes, or multiple inheritances are removed by using interface, etc… These differences between model and OO code make that it is difficult to continue to properly work at a model level as soon as an OO language is used, whereas with an executable class diagram, where the code is added in the methods, the static model remains unchanged: so it is easier to continue to work at a model level.

6 Conclusions and future work

Although work is being done in the area of testing in AOSD [3], little of this work is focusing on the model level [17]. In this paper we have introduced the KerTheme model. KerTheme provides a means for symmetrically decomposing concern based executable class diagrams and concern test scenarios. KerTheme also facilitates synchronised merging of these decomposed models into a coherent composite concern based executable class model and corresponding test scenarios.

In our future work the KerTheme model will allow us to investigate whether decomposed concern based executable class diagrams simplifies the definition of concern test scenarios. This will also allow us to investigate whether this approach ensures more rigorous testing of a complete system. We will evaluate our emerging approach through the Auction Case study described and used in [7] and [8].

Acknowledgements

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] Andrews, A., France, R., Ghosh, S., Craig, G.: Test adequacy criteria for UML design models, Software Testing, Verification and Reliability, 13(2): 95 -127, 2003

[2] Briand, L., Labiche, Y.: A UML-based approach to System Testing. Software and Systems Modeling, 1(1): 10 – 42, 2002

[3] Ceccato, M., Tonella, P., Ricca, F.: Is AOP code easier or harder to test than OOP code?, WTAOP workshop at AOSD '05, Chicago, USA, March 2005

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

[5] Clarke, S., Walker, R. J.: Composition Patterns: An Approach to Designing Reusable Aspects, in Proc: ICSE '01, Toronto, Canada, May 2001.

[6] Dinh-Trong, T., Kawane, N., Ghosh, S., France, R., Andrews A. A.: A Tool-Supported Approach to Testing UML Design Models, in Proc: ICECCS '05, Shanghai, China, June 2005

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

[8] Klein, J., Helouet L., Jézéquel, J.M.: Semantic-based Weaving of Scenarios, in Proc: AOSD '06, Bonn, Germany, March 2006.

[9] Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving executability into object-oriented meta-languages, in Proc: MODELS/UML '05, Montego Bay, Jamaica, October 2005

[10] Muller, P.A., Fleurey, F., Vojtisek, D., Drey, Z., Pollet, D., Fondement, F., Studer, P., Jezequel, J.M.: On Executable Meta-Languages applied to Model Transformations, in MTIP Workshop, Montego Bay, Jamaica, October 2005

[11] Pickin, S., Jézéquel J.M.: Using UML sequence diagrams as basis for a formal test description language. in Proc: IFM '04, Canterbury, Kent, England, 2004

[12] Pickin, S., Jard, C., Le Traon, Y., Jéron, T., Jézéquel J.M., Le Guennec, A.: System Test Synthesis from UML Models of Distributed Software. in Proc: FORTE '02

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

[14] UML Testing Profile, Accessed on: April 2006. http://www.omg.org/docs/formal/05-07-07.pdf [5]

[15] Reddy, R., France, R., Ghosh, S., Fleurey, F., Baudry, B.: Model Composition - A Signature-Based Approach. AOM Workshop AOSD '05, Montego Bay, Jamaica, October 2005

[16] TS, I., ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). 1993, Geneva: ITU-TS

[17] Xu, D., Xu, W.: State-Based Incremental Testing of Aspect-Oriented Programs, in Proc: AOSD '06, Bonn, Germany, March 2006