Compositional Verification of Events and Aspects

Cynthia Disenfeld *

Department of Computer Science
Technion - Israel Institute of Technology
cdisenfe@cs.technion.ac.il

Categories and Subject Descriptors D.2.4 [*Software/Program Verification*]; D.2.13 [*Reusable Software*]: Reuse models

General Terms Languages, Verification

Keywords Aspects, Events, Compositional Verification, Interference, Cooperation

1. Introduction

In AspectJ [9] notation and other aspect languages, the expressive power of pointcuts is restricted. For example, composition is limited to a small set of operators and where and when aspects are woven cannot be defined completely independently: pointcuts are defined in the context of an aspect and advice respond to low-level events.

In [2] these restrictions are explained in a more thorough way together with some previous attempts to solve these issues such as tracematches[1]. Moreover, *events* are introduced, providing a natural solution for the problems mentioned above. Events do not modify the underlying system, but just observe interesting situations and gather information to which other events or aspects may respond. An example of an event in the context of development practices is a commit attempt to the version control repository without having the tests run. Then, an aspect can respond to this event by warning the programmer who is not conforming to the development practices.

My research deals with understanding the implications of introducing language extensions to handle events and developing practical tools and techniques for compositional verification in this context. As a first step of this work, in [4] the issue of event specification and verification was addressed. Different specification languages such as LTL (linear temporal logic), automata, regular expressions and Kripke structures were analyzed. LTL allows expressing properties about the event's exposed parameters according to the events that have occurred before. Other specification languages such as state machines allow expressing properties by means of paths or words accepted, and are particularly useful for defining exactly when the event must or must not be detected.

A compositional technique has been shown where each event should satisfy a specification given by an assumption about the underlying system and a guarantee about the augmented system. This model - consisting of an assumption and a guarantee - is called *assume-guarantee* and has been also used in aspect verification such as MAVEN[6]. The verification techniques involved are mainly model checking, simulation and bisimulation algorithms and static analysis.

Systems usually include several aspects, and weaving all of them into a system may lead to interference. A set of aspects is considered to interfere if each aspect on its own is correct with respect to its specification, but when the whole system is considered, at least one guarantee is no longer satisfied. Interference was considered in [6] for sequential weaving and simple cases of joint-weaving. In the sequential weaving model, an aspect A is woven into a system at the available joinpoints, and then another aspect B can be woven. If B added joinpoints of A, the response is not activated at those joinpoints. In the joint-weaving model, when reaching a joinpoint the corresponding advice is executed, even when it is within the execution of another aspect.

The problem of interference and cooperation among aspects under joint-weaving semantics has been considered in [3] (also part of this research), introducing the specification and verification techniques in order to detect interference or verify the correctness of a set of (possibly collaborative) aspects. The specification now refines the assume-guarantee model, using LTL formulas to express what is expected of the system to be adviced by an aspect A, and what is expected of any aspect to be executed during A. Related work on interference detection is surveyed in [3]. Other work such as [10] assume the existence of interference-detection mechanisms and address the problem of conflict resolution.

* ACM Membership number: 8906706

Copyright is held by the author/owner(s).
AOSD'12, March 25-30, 2012, Potsdam, Germany.
ACM 978-1-4503-1222-6/12/03.

2. Problem formulation

Events and aspects have been considered in [2, 5, 7, 8] but this work focuses on events as defined in [2]. The combination of event and aspect verification is a problem that has not been considered yet and raises new questions such as how the aspect verification technique is affected when aspects respond to events detected. Aspect verification in MAVEN [6] considers pointcuts as temporal logic formulas. Given that events are now more complex and hierarchical entities, event specification should now be used to identify the places where they are detected.

Event guarantees are not necessarily given by an LTL formula. Thus, it is necessary to consider how the model of the aspect assumption incorporates these guarantees in order to be ready to apply aspect verification as in MAVEN. The method obtained should be sound and compositional, in order to allow considering aspects responding to different events, and reusing the proofs in different systems to which the library is applied.

Another question that arises is how to apply compositional interference detection now that aspects may be advised by other aspects because of events detected during their execution. The work already done as part of this research on interference detection [3] will be extended to incorporate events. This implies analyzing new cases of interference and cooperation that may arise from the combination of events and aspects and considering the spectative nature of events. Different event evaluation strategies affect the analysis and obtained results. For example, events that have been detected but no longer hold must be considered in scenarios where aspects might respond to all events detected once in the past. Thus, the possible semantics and their implications will be considered when answering the questions formulated above.

3. Approach

The approach to answer these questions involves combining formal verification techniques. Static analysis can be used for checking that an event does not affect the underlying system, and model checking used to verify that an augmented model satisfies a property. Taking advantage of different techniques' strengths aids in building a simple and natural technique for verifying a library of events and aspects.

An approach to combine these entities (events and aspects) is to consider event guarantees as the pointcut description, so that aspect verification can then be applied. However, aspects might interfere in event guarantees and then interference analysis must be used to make certain that considering event guarantees as pointcuts in aspect verification does indeed yield sound results.

4. Uniqueness and contributions

Verifying a library modularly prevents dependence on a completely built system in order to apply an early detection of bugs and interference. It also allows reusing the library in any other system that satisfies the necessary assumptions without applying any further checks.

The uniqueness of this research is mainly given by building a compositional verification technique that combines events and aspects, allows cooperation, detects interference and is not restricted to sequential weaving semantics, but also accepts aspects that may add and remove events of other aspects as in the joint-weaving model.

Moreover, combining different verification techniques raises new questions as how to get the greatest advantage of each technique and how to combine their results.

5. Results

There are already preliminary results for event specification and verification in [4] and for aspect interference detection and cooperation under joint-weaving semantics in [3].

Providing answers to the problems formulated in Section 2 will provide the user a compositional technique for specifying and verifying a library that includes both events and aspects, understanding how they cooperate or when and why there is interference, and allowing reuse of the library in several different systems.

References

[1] Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Oege De Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding trace matching with free variables to AspectJ. In *OOPSLA '05*.

[2] Christoph Bockisch, Somayeh Malakuti, Mehmet Akşit, and Shmuel Katz. Making aspects natural: events and composition. In *AOSD '11*.

[3] Cynthia Disenfeld and Shmuel Katz. A closer look at aspect interference and cooperation. In *AOSD '12*.

[4] Cynthia Disenfeld and Shmuel Katz. Compositional verification of events and observers: (summary). In *FOAL '11*.

[5] Vaidas Gasiunas, Lucas Satabin, Mira Mezini, Angel Núñez, and Jacques Noyé. EScala: modular event-driven object interactions in Scala. In *AOSD '11*.

[6] Max Goldman, Emilia Katz, and Shmuel Katz. Maven: modular aspect verification and interference analysis. *Form. Methods Syst. Des.*, 37, November 2010.

[7] Adrian Holzer, Lukasz Ziarek, K.R. Jayaram, and Patrick Eugster. Putting events in context: aspects for event-based distributed programming. In *AOSD '11*.

[8] Tetsuo Kamina, Tomoyuki Aotani, and Hidehiko Masuhara. EventCJ: a context-oriented programming language with declarative event-based context transition. In *AOSD '11*.

[9] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. In *ECOOP '01*.

[10] Antoine Marot and Roel Wuyts. Composing aspects with aspects. In *AOSD '10*.