Abstract

Aspect Oriented Programming (AOP) supports the cross-cutting of concerns by means of aspects. The comprehension, maintenance and testing of AO systems may be more difficult than traditional ones, due to the large impact that aspects have on the static structure and dynamic behavior of the overall system.

This thesis proposes the following main contributions to address these open issues: (i) an interprocedural aspect control flow graph (IACFG) representing the interactions among the aspects and the Object Oriented (OO) components of an AO system; (ii) an approach to perform structural testing of aspect oriented programs based on a fault model specific to aspect oriented programs and on a set of coverage criteria referring IACFG; (iii) a metric model and a source code analysis algorithm in order to identify aspects with high level of impact and complexity in AOP systems.

Several experiments were conducted in order to evaluate effectiveness of the proposed approaches.

1. Introduction

Aspect Oriented Programming (AOP) explicitly supports crosscutting concerns by means of program units called aspects [5, 6] encapsulating the crosscutting code. The code in the aspects is able to change both structural (static) properties of the base system and its dynamic behavior at run time. Indeed, by using inter-type declarations in the aspects, new methods and fields can be added to a class of the base system as well as inheritance relationships can be altered and interface implementations forced. Moreover, aspects can define pointcuts specifying both the conditions triggering the execution of blocks of code (advices) in the aspects and the points (join points) in a program execution at which the code in the advices has to be executed. The code encapsulated in aspects, is woven into the code of the traditional program units (e.g. classes in OO code) to build the overall system by the weaving process. The system resulting after weaving presents relationships among aspects and traditional modules that can define a system behavior and structure that may be largely different from the one shown by the base system (i.e. the system with no weaving). This can make the maintenance of a software system implemented by AOP very difficult, or more difficult than a traditional one. Indeed, while on one hand AOP allows to simplify and enhance the design and the code structure (by separating the crosscutting concerns from the principal decomposition and encapsulating them into aspects), on the other hand it could be very difficult to identify and comprehend which are the portions of code directly or indirectly affected by aspects, as well as which are the portions of code that will be actually run at a given point in the code. Although AOP would potentially allow the development of more maintainable systems, if the peculiar AOP constructs are used in undisciplined manner the resulting system could resemble a new kind of spaghetti bowl, with sources of undesired side and ripple effects that are difficult to find and remove. Anyway, also when the AOP constructs are used in a disciplined way, it could be difficult to gain a full comprehension of the system due to the heavy, intrusive, impact that the AOP constructs have on the code they belong. In [2] methodologies to support comprehension, evolution, testing and assessment of aspect oriented software systems are provided. The remaining of the paper is structured as follows. Section 2 briefly presents the IACFG providing a representation of interactions among “aspects” and “classes” The IACFG is also used as foundation for the definition of coverage criteria to perform structural testing, as discussed in section 3, and of a metric model to evaluate the complexities introduced by aspects, as briefly introduced in section 4. Conclusions are finally discussed in section 5.

2. Inter-procedural Aspect Control Flow Graph

The IACFG represents the relationships among classes methods and aspects’ advices. In particular the IACFG shows where the code of an advice is woven
with the code of a method, by highlighting the points in the control flow of an operation where the execution control is transferred to another operation. The IACFG is composed by a set of sub-graphs, the Operation Control Flow Graphs (OCFG), each one representing a (reduced) control flow graph of an operation. A full description of the IACFG and OCFG is provided in [2, 3] along with the rules to recover them by static analysis of the source code. Here, for sake of brevity, it is just worthwhile to note that an OCFG is composed of different kinds of nodes and arcs linking them. A particular type of nodes, the Joint-point shadow Nodes (labeled by JN), highlights the points where the control may be transferred toward an advice, while other types of node indicate the entry (EN) and the end point (FN) of an operation, or traditional code statements (SN), such as assignment, branching, call (C), field reading (FR), field writing (FW), and so on. The XN nodes in the OCFGs, represent the statements where the execution of an operation terminates, such as the return or nodes potentially terminating the execution of an operation because the control transfer towards advices that not proceed. XN nodes are linked to the FN nodes of the OCFG they belong. The pointcut are represented by a syntactic tree of the expression. They are linked by arcs to the OCFG of the advice referring them. An OCFG, corresponding to a given operation, is linked to the OCFG of a different operation, depending on the presence of statements/events transferring the execution control from the first one to the latter. Inter-procedural directed arcs will show the control transfer between operations, i.e. between the corresponding OCFGs. The whole set of OCFGs and the inter-procedural arcs linking them make up the IACFG of the considered system. In particular an inter-procedural directed arc is between a JN node and the entry node of the OCFG corresponding to the advice woven at that joinpoint. The IACFG is the graph resulting by the composition of all the OCFGs linked by the inter-operation arcs. In the IACFG just the OCFGs of operations (directly or indirectly) affected by aspects are reported. The figure 1 reports the IACFG of the example program in figure 2; just the node of interest are reported to represent the interactions among the operations. The algorithm to build the IACFG is based on the (static) analysis of the aspects’ pointcuts expressions. In particular the algorithm inserts join point shadow nodes into the OCFG and adds directed arcs that link the join point shadow nodes, in all the operations, to the advices whose pointcuts picks up them. Some small AO systems were analyzed and represented by the IACFG using the proposed approach with the aim of assessing its feasibility, correctness, and effectiveness, as well as its usefulness in supporting maintenance tasks. A prototype tool was developed to automate most of the approach. The tool, developed using the Eclipse platform, mainly includes: a parser of the AOP code, based on the AJDT parser; the algorithm building the OCFG and IACFG; a graphical utility to visualize the graphs.

3. Structural Testing of AOP source code

A set of testing coverage criteria for AO programs based on the interactions among the advices and the methods has been defined in [2]. The coverage criteria exploit the IACFG and refers to an extension of the fault model by Alexander [1], by considering the AOP unique characteristics and language constructs at a finer grained level. The model is based on a classification of the aspects that considers the interactions among the aspectual and traditional units and takes into account the: (i) the dynamic behavior alterations caused by pointcuts expressions and advice injection; (ii) non-singleton aspect declarations. The focus is on how aspects interact with the classes to alter the behavior or the structure of the system; in particular the following two kinds of alterations have been considered:
include: join point. Failures due to faults in pointcuts or advices, join points and in the advices to be executed at each module boundary can be intercepted, the considered fault model is just based on the interactions among aspects and classes. These interactions are defined by the code specifying pointcuts and advices. Thus faults for these interactions. The IACFG is used as test model representing the interactions among aspects and classes.

- Criterion C1: All Join-point shadow
- Criterion C2: All Join-point shadow branches
- Criterion C3: All pointcut node expressions
- Criterion C4: All pointcut node conditions
- Criterion C5: All pointcut node multiple conditions
- Criterion C6: All Advices

More details about thus criteria are in [2, 4]. Some small AO systems were selected to test them by satisfying the proposed coverage criteria. The aim of the experiments was to assess the validity and effectiveness of the criteria to identify failures in the system under testing, and to support the localization of the faults generating the failures. In [2, 4] are also presented more detail about the results of several analyzed systems.

### 4. A metric model to evaluate aspect impact

The code in the aspects is able to change both structural (static) properties of the base system and its dynamic behavior at run time. This is due to the intrinsic intrusive nature of aspects and the woven process allowing the alteration of both the structure, control and data flow of the components of an AO system. Each of the aspectual constructs can alter the static structure, the control and data flow of the components of an AO system in different way and with a different level of interference. Greater is the level of interference greater is the effort to comprehend the system, lower is the maintainability of the system, higher is the risk of introducing sources for side and ripple effects hard to identify in system maintenance tasks. In [2] a metric model to evaluate the level of interference that aspects can introduce in an AO system. The way an aspect can interfere with the base system by altering its static and dynamic structure have been analysed and classified; interferences causing alterations of (i) the static structure of base components, (ii) the inter-procedural control flow, and (iii) the state of an object have been considered. For each type of these interferences different levels of magnitude have been defined according to the effects they have on the comprehension of the system. The defined scales of interference magnitude allow an evaluation of aspects that is useful to identify critical level...
of interferences in the code, highlighting those components that need to be re-engineered/restructured to reduce the complexity of the overall system. Structural interferences are related to the language constructs affecting the static structure of the base system. Control flow interferences are due to the relationships among class methods and advices that take places at join points, and mainly regard how aspects alter the control flow of original (not-weaved) system. Interferences on the state of objects are due to interactions related to the data that advices can access and eventually modify. Each of these three main categories of aspects interferences may act at a different degree of alterations of the (base) system, going from the absence of structural modifications, or interactions, to a deep invasive alterations affecting the behaviour or the structure of the system. The granularity of such degree levels is a coarse one since all the mainstream AOP languages has adopted a black-box join point model. The figure 3 shows the defined metric model where the where the different interference levels are depicted along three axes each corresponding to one of the three kinds of alterations discussed above. More details are provided in [2]. To verify the effectiveness of the proposed metric model, the AJHotDraw v0.3 system was analysed and the Interference degree Level computed for the aspects in the system. Static source code analysis was performed to identify and classify the relationships among aspects and the components of the base system. In particular the IACFG of the system was exploited to recover the needed information about the kind of the relationships among aspects and the elements of the base system. A chain of prototype tools has been developed and used to support both the IACFG construction and the analysis to identify/classify the aspects interferences.

5. Conclusions

Aspect-oriented languages support new and powerful composition and decomposition constructs that can interfere with the structure of the program making harder to reason about the system’s behavior, program comprehension and maintenance tasks. Novel techniques or adaptation of existing ones are needed to support analysis, testing, quality assessment and evolution of aspect oriented systems. In [2] to address these issues the following contributions have been defined and presented:

- the Inter-procedural Aspect Control Flow Graph to represent interactions among aspects and methods;
- a structural testing approach based on a fault model specific for aspect oriented system
- a metric model and a corresponding program analysis to quantify the impact and complexity of aspects with respect to the base system.

In [2] are also provided experimentation for each of the proposed approach in order to provide an initial validation.

6. Acknowledgments

I would thank my advisor, Prof. Giuseppe Antonio Di Lucca, for his insightful remarks, assistance and suggestions when preparing my Phd. Thesis.

References