TranSAT: A Framework for the Specification of Software Architecture Evolution

Olivier Barais, Eric Cariou, Laurence Duchien,
Nicolas Pessemier, and Lionel Seinturier

Université des Sciences et Technologies de Lille
LIFL, Project INRIA-FUTURS JACQUARD
Fr 59655 Villeneuve d'Ascq
{barais,cariou,duchien,pessemie,seinturi}@lifl.fr

Abstract. Everything changes in our everyday lives: New discoveries, paradigms, styles, and technologies. Frequently, software systems success depends on how they can quickly adapt to requirement or environment evolution. Software architectures are abstract models at the highest level. As such, they should assume conceptual guidance on what parts of the system changed. However, many software architectures often evolve from an uncoordinated build-and-fix attitude. The result is opaque and not analyzable. We present in this paper a practical experience of using aspect oriented programming principles for managing software architecture specification evolution. Our approach aims at clarifying software architecture evolution steps. It extends software architecture abstract models for the specification and the analysis of new concern integration.

1 Introduction

Reusing and integrating heterogeneous software components are one of the major concerns in Component-Based Software Development (CBSD) [9]. Nowadays, industrial component platforms such as CCM [14] or EJB [8], and academic component platforms such as Fractal [7] or ArchJava [1] support the development of distributed applications by assembling components. In these platforms, architecture of an application is a collection of components plus a set of the interactions between them. On the other hand, abstract software architecture models with Architecture Description Languages (ADL) [12] define abstract software architecture models associated with powerful methods and tools. For example, Wright [3] and Darwin [11] support sophisticated analysis and reasoning on architecture properties.

Application evolution grows up software architecture with unexpected functionalities at the first steps of its definition. However previous models and tools are unsuitable for integrating new concerns in already defined software architectures.

With TranSAT (Transform Software Architecture Technologies), we focus on these lacks of evolution definition for software architecture. We propose a

framework for designing a software architecture step by step: From architecture that contains only business concern to a global architecture with business and technical concerns. This framework comes from Aspect Oriented Software Development principles (AOSD) [10] where designers define separately all the facets of an application (business and technical) and then weave them. Indeed in evolution steps, software architects should integrate new components to provide new features. AOSD provides solutions to weave them. In TranSAT, we propose a point-cut mask that forces the architecture transformation.

In this paper, we first provide an overview of TranSAT framework. Then, in section 3, we detail the concept of *point-cut mask* that ensures a safe integration of some concerns in architectures. Finally, in section 4, we present a short conclusion of our work and open issues.

2 TranSAT: A framework for modelling software architecture evolution

2.1 Basic software architecture model

Software architecture research community focuses on building formal notations in order to define structure and behaviour of software architecture. They are recognized in Architecture Description Language (ADL) domain. Nowadays, several ADLs as Wright [3] or Darwin [11] are mature. They become effective vehicles for communication and analysis of a software system. However the integration of these specification languages in an iterative process keeps being difficult.

In TranSAT, we propose adding AOSD principles to support separation of concerns and evolution in architecture context. First, we build our own software architecture model called SafArchie [6][5]. SafArchie provides a light and hierarchical component model. A component is defined by its structural interface but also by its external behaviour specification. The structural interfaces are a set of *operation* prototype gathered within *Ports*. The external behavior specification defines the trace of the messages that the component sends or receives. SafArchie architectural specifications can be analyzed and transformed in Fractal or ArchJava specification. Our component model comes from ArchJava component model [2]. Therewith, we add an external behavioural specification by a subset of FSP language used in Darwin for analyzing some composition properties.

2.2 TranSAT overview

Various works on software engineering note the difficulty to think about all the concerns of software in the design of large architectures. The separation of concern approach, mainly developed by Kizcales [10] and implemented in AspectJ [4], can be applied to software architecture approach to adapt an architecture specification. TranSAT is a framework that refines software architecture specifications by adding technical concerns such as persistence, security or

transaction management. Integration of these concerns acts upon the software architecture specification by architectural, structural, and behavioural changes. The modularity principle implies several difficulties:

- Concerns must be defined independently from software architecture specifications to be easily reused.
- Most of the time, a technical concern can modify several business components, it *crosscuts* them. For example a security concern can modify the structure and the behaviour of interaction partners.
- The integration rules of a concern on a software architecture should be specified, analyzed, and saved.

For solving these issues, on top of our component model, TranSAT adds two new concepts (see Fig 1).

First, the ***adapter*** defines the integration rules of technical components on a generic software architecture. It is unaware of the integration context. For example, it specifies the integration rules of a security concern.

Second, the ***weaver*** describes the interaction between a specific software architecture and a technical concern. It binds for example a security concern with gas station software architecture. It identifies precisely the interaction space where the software architecture specification is updated by a point-cut. For keeping AOSD principles, we define point-cut as a set of hooks on a software architecture. In our software architecture model, those hooks are before, after, or around an operation. Our approach is leaded by a multiple actor view. In large software engineering project, different actors play specific roles. TranSAT approach identifies four main roles: developer, analyst, integrator, and architect. The architecture specification from three main concepts (components, adapters, and weavers) clearly separates their work space. With TranSAT, the new architect role consists of assembling some components but also weaving them with new concerns.

Point-cut definition

A technical concern service is defined by a set of components and their interactions. Its integration modifies the target architecture. We specify its integration with two steps. First, we declare the integration rules independently of the context. Secondly, we configure this integration for a specific target software architecture.

The main difficulty in a weaving model consists in providing a powerful language to identify where a concern hooks on a software architecture. In Aspect Oriented Programming language, point-cut definition identifies this space. In TranSAT, interaction space between components and concerns has a three level definition (see Figure 2). One of them, named *point-cut* definition, consists of a hook identification for target software architecture. In the adapter we generalize this definition with a *point-cut label* definition. Point-cut label is identified by a name. It serves as reference for the integration rules specification. It is associated with a *point-cut mask* that defined a set of structural, behavioural, and architectural constraints on a software architecture model (see section 3). Therefore

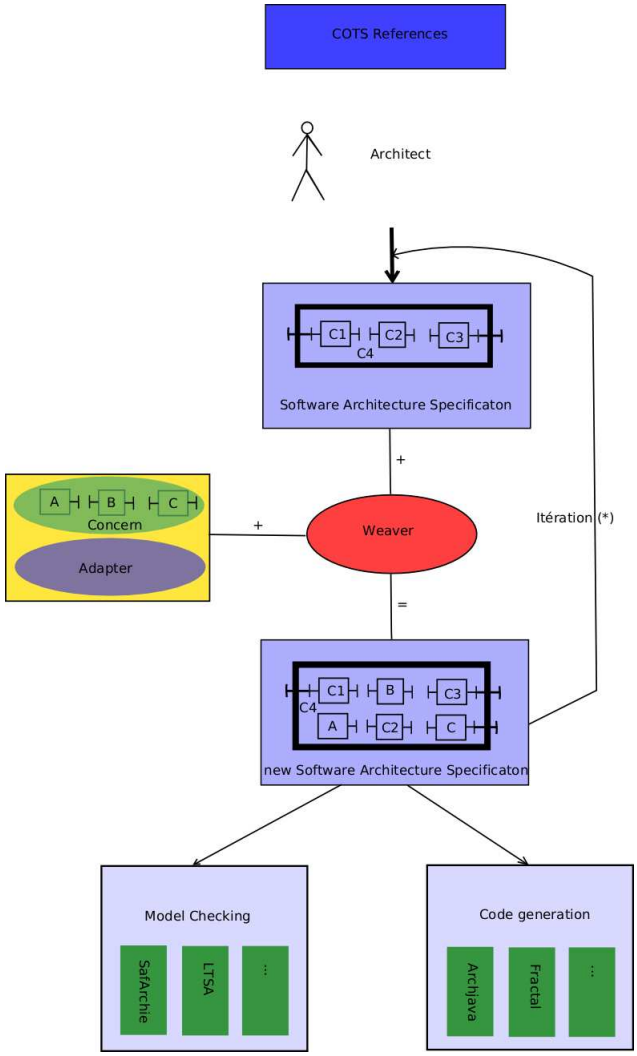

Fig. 1. TranSAT overview

interaction space identification can be compared to a variable definition in programming languages. Like a variable, an interaction space is named (point-cut label), typed (point-cut mask), and instantiated for a specific context (point-cut definition).

Adapter: Saving context independent integration rules
In TranSAT, adapter contains integration rules of a concern on a generic software architecture. We define three kinds of integration rules: structural, behavioural, and architectural transformation rules. The latter updates interactions between

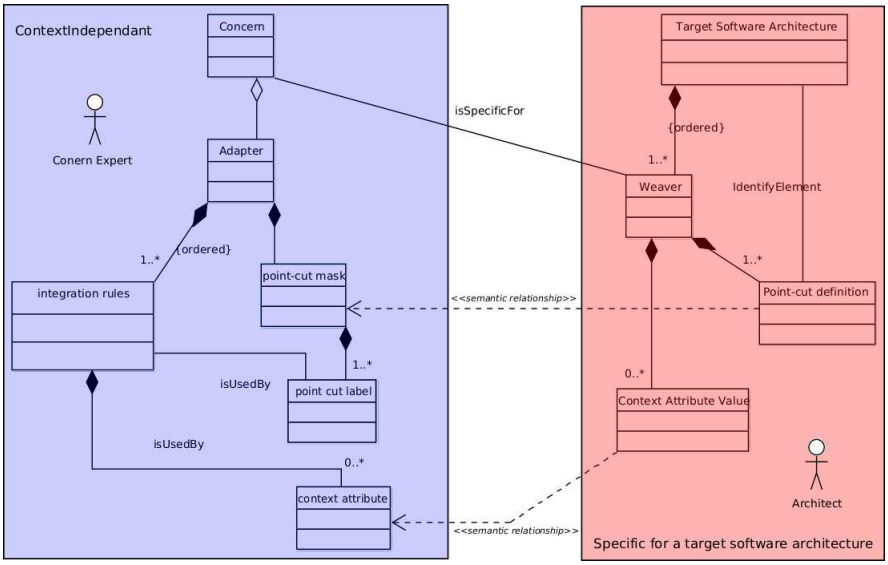

Fig. 2. TranSAT meta-model

components. They are defined from point-cut labels (see above). These integration rules are context independent. Adapter needs information to achieve the transformation. That information is defined in adapter's configuration interface (*context attribute* in figure 2). They are given at the weaving step (*context attribute value*).

Transformation specification assumes hypothesis on software architecture. They are specified within the point-cut mask detailed in section 3. The conformity between a point-cut mask and a point-cut definition are checked at the weaving step.

Weaver: Integration configuration for a target software architecture
To specify software evolution, architect first chooses a concern and its adapter. Secondly, he/she defines point-cut label on the target software architecture. Thirdly, he/she gives some contextual information expected by adapter to achieve this transformation. Finally, he/she links point-cut definitions with point-cut labels. This link represents the weaving between a set of transformation rules and target software architecture.

The weaver gathers the information specified by architect for the integration of a technical concern on specific software architecture. Several steps of evolution could be defined in order to add different technical concerns. Each step is performed by a specific weaver. The architect specifies manually the processing order of the weavers. Each evolution step defines new component based software architecture.

3 The point-cut mask

3.1 Goal

A point-cut mask is a contract for a point-cut definition that ensures the correct integration of a concern on software architecture. The adapter defines a point-cut label from which the transformation process is specified. The point-cut mask compels the point-cut definition to respect limits in its definitions. It specifies structural, behavioural, and architectural constraints that are expected by the adapter and that should be assumed by the target software architecture for the integration of a concern. We check in our integration process that a point-cut definition respects the contract defined by its point-cut mask.

Point-cut mask and point-cut definition have not the same role. Although integrators define the point-cut mask in the adapter for a set of integration rules, architect identifies the interaction space where a target software architecture specification is updated with the point-cut definition. The point-cut mask is context independent although the point-cut definition is specific for the integration of one concern on software architecture.

3.2 Proposition

The point-cut mask specifies a set of structural, behavioural, and architectural constraints. For an architecture evolution specification, the new architecture should assume those constraints.

The point-cut mask definition is a software architecture specification, but it does not need to be complete, i.e. several elements of software architecture can be undefined or specified with regular expression. A point-cut mask specification respects the meta-model presented figure 3.

In figure 4, the point-cut mask is the following for internationalization concern integration. A point-cut definition will be compliant to this point-cut mask and then will contain a string to translate. In this example, as the translation is a simple indirection, there is no hypothesis on component external behaviour and no hypothesis on architecture configuration.

In design by contract approach [13], a precondition is an assertion that must be true before a method invocation. If we consider the evolution step as an atomic operation, point-cut mask can be compared to a transformation pre-condition. Point-cut mask must match with point-cut definition on specific software architecture. This contract improves the transformation consistency. It ensures balance between the software architecture model expected by the adapter and the point-cut on specific software architecture.

4 Conclusion and open issues

This paper highlights our research in architectural evolution and transformation consistency checking. As part of architectural evolution, we define a three-view approach: the definition of concern interface, the integration rules specification, and the integration configuration. Using AOP principles in software architecture specification is an interesting approach. Indeed, several component based platforms provide hooks for the weaving of concerns [7] [15]. We use the same concepts to identify the interaction space where a target software architecture

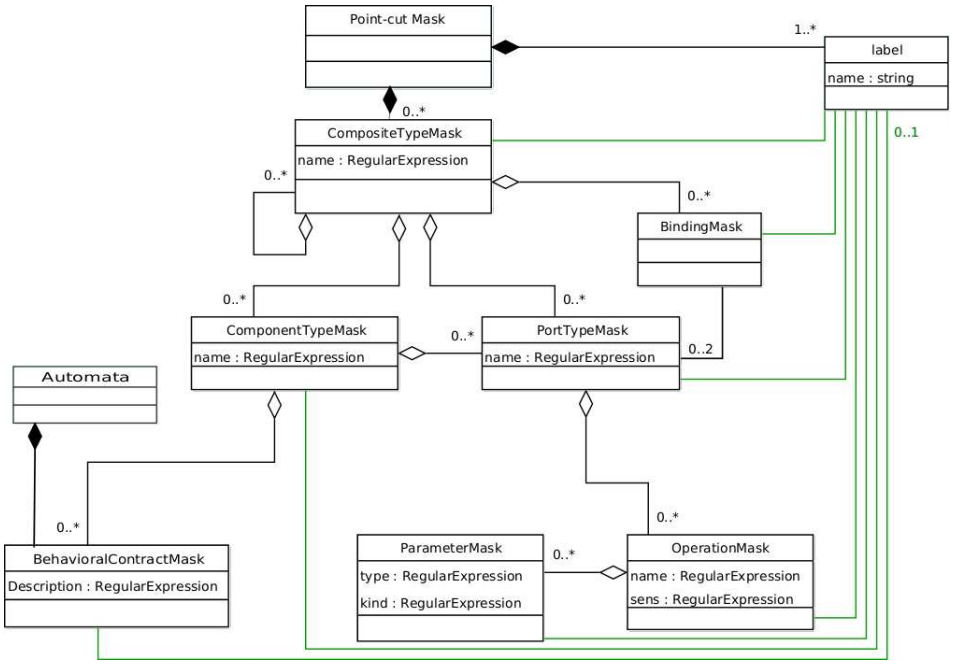

Fig. 3. Point-cut mask meta-model

```
<ComponentMask Name="*">
    <PortMask Name="*">
        <OperationMask Name="*" ProvidedRequired="*" Pointcut="true">
            <ParameterMask Place="*" Kind="*" Type="String"/>
        </OperationMask>
    </PortMask>
</ComponentMask>
```

Fig. 4. Point-cut mask example

specification is modified. Therefore this approach defines an abstraction for software architecture evolution that could be analyzed.

We have tooled some concepts discussed in this paper and extended our architecture tool suite: SafArchie Studio. This tool is a set of new extensions for ArgoUML. It proposes several views of software evolution for developers, analysts, integrators, or architects.

TranSAT approach is in progress. Point-cut mask is a first level of information to ensure software architecture evolution with a consistency check. This latter is only carried out before the transformation. Next steps consist in understanding which kinds of contract can be expressed as invariant or post-condition for a software architecture evolution. Those new contracts should provide architects methods for the evolution impact analysis.

References

1. J. Aldrich, C. Chambers, and D. Notkin, *Architectural Reasoning in Archjava*, Proceedings ECOOP 2002 (Malaga, Spain), LNCS, vol. 2374, Springer Verlag, June 2002, pp. 334–367.
2. ———, *ArchJava: Connecting Software Architecture to Implementation*, Proceedings of the 24th International Conference on Software Engineering (ICSE-02) (New York), ACM Press, May 19–25 2002, pp. 187–197.
3. R. Allen, *A Formal Approach to Software Architecture*, Ph.D. thesis, Carnegie Mellon, School of Computer Science, Janvier 1997, Issued as CMU Technical Report CMU-CS-97-144.
4. AspectJ Team, *The AspectJ programming guide*, Available from http://aspectj.org/doc/dist/progguide/index.html, February 2002.
5. O. Barais and L. Duchien, *Safarchie studio: Argouml extensions to build safe architectures*, Workshop on Architecture Description Languages, IFIP WCC World-Computer Congress (Toulouse, France), 2004.
6. O. Barais, L. Duchien, and R. Pawlak, *Separation of Concerns in Software Modeling: A Framework for Software Architecture Transformation*, IASTED International Conference on Software Engineering Applications (SEA) (Los Angeles, USA), ACTA Press, november 2003, *ISBN 0-88986-394-6, pp. 663–668.*
7. *E. Bruneton, T. Coupaye, and J.B. Stefani,* The Fractal Component Model, version 2.0-3*, Online documentation http://fractal.objectweb.org/specification/, February 2004.*
8. *L.G. DeMichiel,* Entreprise Javabeans Specification, version 2.1*, Online documentation http://java.sun.com/products/ejb/docs.html, June 2002.*
9. *G. Heineman and W. Councill (eds.),* Component-Based Software Engineering, Putting the Pieces Together*, Addison-Westley, 2001,* ISBN: 0-201-70485-4.
10. *G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J-M. Loingtier, and J. Irwin.,* Aspect-Oriented Programming*, Proceedings ECOOP (Mehmet Akşit and Satoshi Matsuoka, eds.), vol. 1241, Springer-Verlag, 1997, pp. 220–242.*
11. *J. Magee,* Behavioral Analysis of Software Architectures using LTSA*, Proceedings of the 21st international conference on Software engineering, IEEE Computer Society Press, 1999, pp. 634–637.*
12. *N. Medvidovic and R. N. Taylor,* A classification and comparison framework for software architecture description languages*, IEEE Transactions on Software Engineering, vol. 26, Janvier 2000.*
13. *B. Meyer,* Applying "Design by Contract"*, Computer* ***25*** *(1992), no. 10, 40–51.*
14. *OMG,* CORBA Component Model, v3.0*, Online documentation http://www.omg.org, June 2002.*
15. *R. Pawlak, L. Seinturier, L. Duchien, and G. Florin,* JAC: A flexible solution for Aspect-Oriented Programming in Java*, Lecture Notes in Computer Science* ***2192*** *(2001), 1–24.*