Model Checking the Behavior of Frameworks Extended with Other Frameworks

Federico Balaguer
University of Illinois Urbana Champaign
201 North Goodwin Ave.
Urbana, IL - USA
balaguer@uiuc.edu

ABSTRACT

Frameworks are important in software development. There are problematic aspects of framework development. When frameworks are extended with functionality implemented by other frameworks, developers face a difficult task solving static and (specially) dynamic mismatches. The dynamic aspect is less visible to developers thus it is usually the cause of failure, specially in frameworks using multi threading programming.

This paper shows how to verify the soundness of framework compositions at dynamic level using using temporal logic and tools provided by Full Maude.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*model checking*

General Terms

Design, Verification

Keywords

Object-Oriented, Frameworks, Rewriting Logic, Temporal Logic

1. INTRODUCTION

Frameworks have become popular in the object-oriented community [9]. SourceForge has at least 600 projects in which the deliverable is a framework [1]. However, framework-based development is not a mature discipline [6]. Traditional software-engineering techniques do not handle frameworks very well because frameworks are different from traditional object-oriented architectures. Frameworks have two main characteristics. First, frameworks are incomplete designs [12]. Users of the framework have to complete the design of the framework with their own code. Second, the framework defines a flow-of-control that is independent from the flow of control of the programs where the framework is instantiated.

When a framework needs to be extended with new functionality, developers can choose between implementing the new functionality from scratch or using an existing framework on the required domain. In theory, the later is more attractive because it is reusing the knowledge of the domain provided by the extending framework. In reality, developers have to solve different kind of mismatches at the architecture level [5] or the API level [8]. Moreover, after solving these mismatches the extended framework could have two problems. First, previous instantiations of the framework are no longer valid. Second, the behavior is incorrect [4].

We claim that there is a systematic approach to extending frameworks that allows developers to understand and to evaluate the composition as another software engineering artifact [4]. This approach takes in consideration how each framework is instantiated, extended, and how is configured at run-time [3]. A new version of the framework is defined as the instantiation of the framework supplying the behavior and the extension of the framework that needs the new functionality. The flow of control of the extended framework is defined as the composition of the flow of controls of the extending and extended framework.

This paper presents the results of using Full Maude on describing and composing two object-oriented frameworks. We used Maude's LTL checker [10] for asserting desirable properties of two frameworks and also for checking that their composition is sound.

2. OBJECT-ORIENTED FRAMEWORK AS REWRITE THEORIES

A rewrite theory is a triplet $\mathcal{R} = (\Sigma, E, R)$, with (Σ, E) an equational theory, and R a set of labeled rewrites rules that are applied modulo the equations E [14]. Maude and Full Maude provide modules that allow the user to create object-oriented specifications [7]. In Full Maude, object-oriented modules implicitly include the declaration of sorts: Oid (object identifiers), Cid (class identifiers), Object (instances of classes) and Msg (sent messages). Instances and messages created by one specification form a CONFIGURATION, it is sometimes called "soup".

A framework can be defined with a rewrite theory $\mathcal{R}_f$ as a triple, $\mathcal{R}_f = (\Sigma, E, R)$ as for any rewrite theory. The distinction is that neither E nor R are complete. There are either missing equations in E or missing rewrite rules on R. Developers customize the framework in two different ways.

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

One way, they add missing equations or rewrite rules (white-box). Another way, they create an initial configuration with objects conforming to a given API (black-box) [13].

Two frameworks were developed [2] with the object-oriented extension of Full-Maude [7]. One framework (Channels) allows developers to create channels that a collection of Senders use to broadcast data to receivers. The other framework allows to create Pipe&Filter architectures. For each framework we had defined a number of temporal logic assertions that help us distinguish between good and bad behaving instantiations. For example, we wanted to specify that a Sender in the Channel framework is always able to send all its data:

```
eq( <A:Sender | buff:nil> C ) |= sendAll = true .
...
eq fairToSender = <>[] sendAll .
```

Note that this allows us to still define faulty channels that lost data. For the Pipe&Filter framework we wanted to specify that data push in the front-end of the Pipe produces an output in the back-end:

```
eq( <A:OutPipe | ready:false> ) |= dataOut = true .
...
eq outReady = <>[] dataOut .
```

These equations are defined in modules that wrap the modules with the framework specification.

3. CHANNEL WITH PIPES&FILTERS

Imagine that we need to implement a new type of Channel that performs a series of operations on new entered values to the buffer. One possibility could be to write a new subclass that does it from scratch. The other possibility is to create a new channel that uses the Pipes&Filters framework to implement the transformations. The advantage of reusing the Pipes&Filters framework is that we are reusing all the domain knowledge and infrastructure that comes with it.

The Channel framework is extended by creating a new subclass of Channel with the required functionality. In this case the new class (we called PipeChannel) will create a configuration out of the Pipe & Filter frameworks. Once the static aspect of the extension is developed, the extended framework can be Model Checked using temporal logic predicates. For example, in the new extension should be true:

```
[]( fairToSender /\ outReady /\ ... ).
```

It can also be tested using the Search tool that explores all possible resulting configurations that can be found from the original state.

4. CONCLUSIONS AND FUTURE WORK

Framework composition is still an unexplored area in software engineering. Framework composition is usually developed based on a succession of trials without knowing how or why things work. Using testing packages such as SUnit or JUnit is just a more sophisticated way of doing the trials. A better understanding of each framework design is required. Our goal is to provide tools and techniques that ease the development and evaluation of this kind of framework development.

This work shows the benefit of using executable specifications for verifying the correctness of object-oriented frameworks. It shows how even predicates defined in the abstract and incomplete design of a framework can be successfully applied to instantiations of frameworks.

This work also shows that it is not trivial to develop a specification that later can be model check. Moreover, anyone doing such specification has the pending obligation to prove that the specification reflects the behavior of the targeted system (in our case the extended frameworks). One possible approach is to have an interpreter for the language in which the frameworks are implemented and then use the interpreter to load the source code of the frameworks to perform the validation. Although this approach is conceptually simple, it has the problem of having to load the classes of the library to which the framework depends on. Currently we are developing a different approach in which flow-of-control are abstracted out of the framework (as part of the documentation). An interpreter of the flow-of-control is built in Maude to Model Check different properties of each framework.

5. REFERENCES

[1] Sourceforge. http://sourceforge.net.

[2] F. Balaguer. Channels and pipes & filters (source code). "netfiles.uiuc.edu/balaguer/wwww/maude".

[3] F. Balaguer. Design aspects for describing frameworks. In *Companion of Conference on Object-Oriented Programming Systems, Languages and Applications*, 2001.

[4] F. Balaguer and R. Johnson. Composing frameworks by separating concerns. In *Conference on Software Engineering and Applications (SEA'03)*, 2003.

[5] L. Berlin. When objects collide: Experiences with reusing multiple class hierarchies. In *Conference on Object Oriented Programming, System, Languages and Applications*, 1990.

[6] J. Bosch, P. Molin, M. Mattson, P. Bengtsson, and M. Fayad. *Framework Problems and Experiences.* In Fayad et al. [11], 1999.

[7] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *Maude 2.0 Manual.* SRI International, June 2003.

[8] R. A. David Garlan and J. Ockerbloom. Architectural mismatch: Why reuse is so hard. *IEEE Software*, 1994.

[9] D. D'Souza and A. C. Wills. *Objects, Components and Frameworks with UML. The Catalysis Approach.* Addison-Wesley, 1999.

[10] S. Eker, J. Meseguer, and A. Sridharanarayanan. The maude ltl model checker. In *4th International Workshop on Rewriting Logic and its Applications (WRLA'02)*, 2002.

[11] M. Fayad, D. Schimidt, and R. Johnson, editors. *Object-oriented Foundations of Framework Design.* . Willey, 1999.

[12] R. Johnson. Frameworks = components + patterns. *Communications of the ACM*, 40(10):39–42, October 1997.

[13] R. Johnson and B. Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, June/July 1988.

[14] J. Meseguer. Software specification and verification in rewriting logic. Technical report, Lectures at the Marktoberdorf International Summer School, Germany.