A Multi-Stage Approach for Reliable Dynamic Reconfigurations of Component-Based Systems[*]

Pierre-Charles David[1], Marc Léger[2], Hervé Grall[1], Thomas Ledoux[1], and Thierry Coupaye[2]

[1] OBASCO Group, EMN / INRIA, Lina
École des Mines de Nantes
4 rue Alfred Kastler
F-44307 Nantes CEDEX 3
[2] France Télécom, Recherche & Développement
28, chemin du vieux chêne
F-38243 Meylan

Abstract. In this paper we present an end-to-end solution to define and execute reliable dynamic reconfigurations of open component-based systems while guaranteeing their continuity of service. It uses a multi-stage approach in order to deal with the different kinds of possible errors in the most appropriate way; in particular, the goal is to detect errors as early as possible to minimize their impact on the target system. Reconfigurations are expressed in a restricted, domain-specific language in order to allow different levels of static and dynamic validation, thus detecting errors before executing the reconfiguration where possible. For errors that can not be detected early (including software and hardware faults), a runtime environment provides transactional semantics to the reconfigurations.

1 Introduction

Complex software systems must be modified/maintained during their lifetime, for example to fix bugs or include new functionalities. It is often not practical – or even possible – to stop the system in order to perform these changes. Instead, the changes must be applied dynamically to keep the running system available.

There are two conflicting forces that make evolution especially challenging. On the one hand, the evolutions that will be applied to a system cannot be precisely anticipated at the time it is initially built and deployed. This means the system must be kept open and flexible to accommodate future needs. On the other hand, modifying production systems that are often business-critical is very risky, and we need to ensure that these changes will cause the minimum possible disruption, even though we do not know ahead of time the actual changes that

will be made. In short, we need a way to provide *reliable dynamic reconfigurations*. By *reliable* we mean: *(i)* reducing as much as possible the occurrence of errors (fault prevention), *(ii)* when errors that could not be prevented actually happen, minimize the damage they cause to the system (fault tolerance).

This paper introduces a modular validation chain to support *reliable* dynamic reconfigurations on top of general-purpose component models like Fractal [1]. The chain is based on a decomposition of the life-cycle of individual reconfigurations in multiple stages, from their definition to their actual execution on the target system. As reconfiguration scripts go through these successive stages, different techniques are used to "weed out" incorrect reconfigurations and handle errors that could not be prevented. The different stages of the validation chain complement each other to offer strong reliability guarantees. At the same time, the chain stays modular and can be customized to support different tradeoffs between performance and guarantees depending on the domain.

In the rest of this paper, we first present the overall architecture of our approach (Sect. 2), detailing the different kinds of errors our validation chain handles thanks to its multi-stage architecture. Sections 3 to 5 then give more details on each successive stage of the chain. We conclude (Sect. 6) with an overview of the current status of these different modules and of the future work.

2   Overview of the Validation Chain

The goal of the proposed validation chain is to ensure the reliability of the dynamic reconfigurations of software architectures (e.g. component replacement, reconnection of component bindings). It relies on the use of a dynamic component model that supports unanticipated reconfigurations, typically thanks to reflective features. In our case we use Fractal [1] for its flexibility and extensibility.

The main idea behind our proposal is to handle the different kinds of errors at different points in the life-cycle of a reconfiguration. Accordingly, the validation chain is organized with three main stages as illustrated in figure 1, each stage corresponding to a different step in the life-cycle of the scripts that describe the reconfigurations to be executed. At each stage, if an error is detected, the reconfiguration is immediately rejected. Hence the whole chain acts as a sequence of increasingly specific sieves that scripts must pass through, from basic sanity checks to a full-blown managed execution of the reconfigurations as transactions.

Loading. First, the specification of a reconfiguration is loaded into the validation chain, in the form of a *reconfiguration script*. Such a script can be executed many times in its lifetime, with different target architectures, but it is loaded only once. For example, a generic component replacement script can be reused with different parameters each time a component must be updated to a new version. At this time, the possible target architecture are only specified by the architecture model, which defines some rules that the architectures under consideration satisfy. With these informations, the possible validations include various levels of *static analyses* filtering out reconfiguration scripts that could cause errors when applied to a concrete architecture.
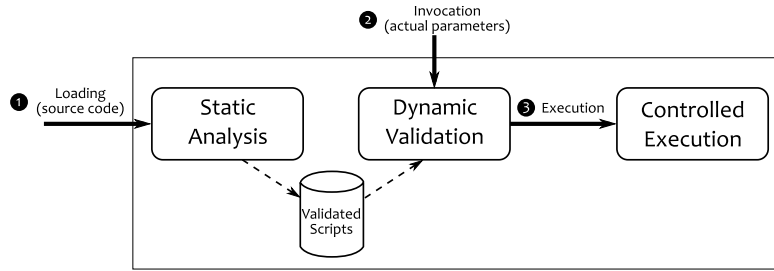
Fig. 1. The validation chain's architecture

Invocation. After loading, a user may want to actually execute the reconfiguration on a particular target architecture. Some additional validations can now be performed: this stage filters out scripts that are incompatible with the given target architecture.

Execution. Finally, the reconfiguration is executed on the target architecture. If the previous steps have been precise enough, most erroneous reconfigurations have already been rejected at this point. However, some kinds of errors are either impossible to predict (e.g. hardware faults) or too costly to detect. To handle these errors, the execution stage uses a runtime environment providing transactional properties to reconfigurations in the Fractal model. Although it can actually handle all the errors detected by earlier stages, this choice may make the architecture not available during a too long time.

The different stages of the validation chain work together providing an integrated whole. At the same time, the chain stays modular, and some of the stages can be disabled or replaced. As the different analysis techniques have different costs, the validation chain can be customized depending on the target architectures: critical systems will require more complex static analyses in the earlier stages, and may even include a test run on a replica system whereas the cost of these steps may be redhibitory in other contexts. The rest of the paper gives more details on each of the successive chain stages.

3   Static Analysis With Respect to the Architecture Model

The first stage in the validation chain loads the source code of the reconfiguration script into the chain. Its goal is to verify the validity of the reconfiguration with respect to the underlying architecture model. The component model defines some rules to be satisfied by the architectures under consideration. At this point, the actual architectures to which the script will be applied are unknown.

The reconfiguration scripts are written in a domain-specific language named FScript [2] that we have defined for this purpose. This language not only allows reconfigurations to be easily expressed, but also ensures some safety properties: for instance, any well-formed script terminates.

When a script is well-formed, a semantic analysis introduces an axiomatic definition of the script execution. This analysis is parameterized by a selection of the rules of the architecture model: this allows us to easily support variants, at the infrastructure or application levels, like different architectural styles. Note that some rules can be discarded because they are too costly to analyze. The analysis defines Hoare's correctness formulas $\{P\}S\{Q\}$ where S is the script and P and Q are properties describing the architecture to be reconfigured (expressed in first-order logic). Such a formula means that any architecture satisfying the precondition P will satisfy the postcondition Q after the completion of the script S. The aim of the semantic analysis is to determine a precondition P that does not lead to an error state where the architecture violates some invariant rules under consideration: only the architectures that satisfy the precondition P will be reconfigured by the script. Therefore, if the precondition P is *false*, it means that the script is not useful according to the analysis and should be rejected. Otherwise, the script is considered as potentially valid, and passed to the next stage of the validation chain along with the computed precondition. Note that the semantic analysis can be more or less precise: the precondition P is sufficient to ensure the absence of errors with respect to the selected rules, but not necessary.

4 Validation With Respect to the Target Architecture

The second stage in the validation chain is triggered each time the user requests the invocation of a reconfiguration script by giving a target architecture and actual parameters. This stage performs additional validations thanks to this information, but without actually modifying the target architecture.

At this point, the script has already passed the first stage of the chain, and has a pre-condition associated to it. The first step is thus a simple *compatibility check*, which consists in evaluating the pre-condition on the target architecture and the actual parameters. This can be done easily, and only requires to introspect the target architecture, without modifications.

If the compatibility check has succeeded, an optional second step can be included, which consists in a *simulation* of the script's execution. This step uses a virtual implementation of the target architecture, on which the reconfiguration script is executed using the script interpreter. The virtual architecture is initialized with the initial state of the target system, but implements "copy-on-write" semantics: operations are applied to the virtual copy, and do not modify the actual target system. If any of the component model invariant rules of the architectural model are violated during the simulation, the invocation is rejected.

One advantage of the simulation is that it can be more precise (and thus can catch more errors) that the static analyses, which may be restricted by a selection of the invariant rules to be preserved. Also, by instrumenting the virtual architecture to be reconfigured, it can generate the exact trace of the reconfiguration performed by the script, which can be "replayed" with very little overhead to reproduce its effect on the actual target system [3]. The only drawback is that this step can increase the latency of the reconfiguration.

5   Execution of Reconfigurations as Transactions

The final stage of the chain is the actual application of the reconfiguration script
on the target architecture. Depending on how the previous stage was config-
ured, it uses either a compiled form of the script, or the specialized trace of
reconfiguration operations that was generated during the simulation.

Because the overall objective of the validation chain is to guarantee the re-
liability of the reconfiguration, this step must either apply the complete recon-
figuration script without errors, or, in case of errors, restore the system to the
last consistent state before the execution of the script by rollbacking the failed
reconfiguration: it must be fault tolerant. The failures that happen during the
actual execution of the reconfiguration include software failures (e.g. violation
of the architecture model) that were not detected earlier, and some errors that
are fundamentally impossible to predict (e.g. hardware crashes). In all cases, the
resulting architecture must be in a consistent state according to the definition
of the underlying architecture model.

These objectives call for the use of transaction management techniques, as
they closely match the standard ACID properties (Atomicity, Consistency, Isola-
tion, Durability) of transactions in distributed computing [4]. In order to execute
reconfiguration scripts inside global transactions with automatic demarcation,
we use an extended version of the Fractal component model [5], which provides
transactional semantics for Fractal architectures. Therefore, reconfigurations can
benefit from ACID properties to support concurrency, recovery, and to guarantee
system consistency.

6   Conclusion and Future Work

The objective of this work is to make runtime reconfigurations of open software
architectures reliable while maximizing their availability. We specially target
reflexive component-based architectures for their suitable adaptability property,
as exemplified by the use of the Fractal model in our current implementation.
Our solution relies on a multi-stage validation chain with two main dependability
methods: fault prevention and fault tolerance. Fault prevention notably includes
the use of static analysis on a dedicated reconfiguration language in order to
detect invalid reconfigurations with respect to the architecture model, and an
additional simulation stage on the target architecture. Fault-tolerance is ensured
by a transactional runtime for the actual execution of reconfigurations.

Although several component models support open dynamic reconfigurations,
they do no take into account the reliability of reconfigurations. On the contrary,
most work on reliability and validation for component-based architectures deal
with Architecture Description Languages [6,7] only include static validations
and do not support unanticipated reconfigurations. Recent component models,
like FORMAware [8] and Plastik [9], rely on reflexive architectures to allow unan-
ticipated reconfigurations while supporting some kinds of guarantees checked at
runtime. Our work differs in that we provide a multi-stage architecture that

integrates different complementary validation techniques in a consistent whole. Depending on the domain requirements, the focus of the validation chain can be put on the static validation, the controlled execution, or both, for instance for critical systems.

Currently the overall architecture of the validation chain is in place, and the whole system is usable although some of the individual stages are not yet complete: the simulation and execution of reconfiguration programs is fully functional, including transactional guarantees. Our current focus is on the earlier steps, and in particular the definition and implementation of the static analysis of reconfiguration scripts, which requires a formal definition of both the FScript language and the Fractal model. Once we have a fully implemented validation chain for Fractal, we plan to extend it to support other component models.

References

1. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.B.: The Fractal Component Model and its Support in Java. Software Practice and Experience, special issue on Experiences with Auto-adaptive and Reconfigurable Systems **36**(11-12) (2006) 1257–1284
2. David, P.C., Ledoux, T.: Safe dynamic reconfigurations of Fractal architectures with FScript. In: Proceedings of the 5th Fractal Workshop at ECOOP 2006, Nantes, France (July 2006)
3. Polakovic, J., Mazaré, S., Stefani, J.B., David, P.C.: Experience with implementing safe reconfigurations in component-based embedded systems. In: The 10th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE 2007). LNCS, Springer Verlag (July 2007)
4. Traiger, I.L., Gray, J., Galtieri, C.A., Lindsay, B.G.: Transactions and consistency in distributed database systems. ACM Trans. Database Syst. **7**(3) (1982) 323–342
5. Léger, M., Ledoux, T., Coupaye, T.: Reliable dynamic reconfigurations in the Fractal component model. In: Proceedings of the 6th workshop on Adaptive and reflective middleware (ARM'07), New York, NY, USA, ACM (2007) 6
6. Allen, R.J.: A Formal Approach to Software Architecture. PhD thesis, Carnegie Mellon University (May 1997) Technical Report Number: CMU-CS-97-144.
7. Medvidovic, N., Oreizy, P., Robbins, J.E., Taylor, R.N.: Using object-oriented typing to support architectural design in the C2 style. In: Proceedings of the ACM SIGSOFT'96 Fourth Symposium on the Foundations of Software Engineering, San Francisco, CA, USA, ACM SIGSOFT (October 1996) 24–32
8. Moreira, R.S., Blair, G.S., Carrapatoso, E.: Supporting adaptable distributed systems with FORMAware. In: ICDCSW '04: Proceedings of the 24th International Conference on Distributed Computing Systems Workshops, Washington, DC, USA, IEEE Computer Society (2004) 320–325
9. Batista, T., Joolia, A., Coulson, G.: Managing dynamic reconfiguration in component-based systems. In: 2nd European Workshop on Software Architectures (EWSA 2005). (2005)