

Architecture Recovery for Distributed Systems

Nabor C. Mendonça
Centro de Ciências Tecnológicas
Universidade de Fortaleza
Av. Washington Soares, 1321, Bloco D
60811-905 Fortaleza, CE, Brazil
nabor@unifor.br

Abstract

The ability to recover up-to-date architectural information from existing software artifacts is key to effective software maintenance, reengineering and reuse. Although architecture recovery can be facilitated with the help of current reverse engineering techniques and tools, many issues remain to be properly addressed, particularly regarding recovery of runtime abstractions (e.g., clients, server, interaction protocols) that are typical to distributed system design. In this position paper we claim that integrating multiple static analysis techniques can be a cost-effective way of recovering a distributed system's "as-built" architectural design. To support this claim, we present X-ray, an integrated, static analysis based architecture recovery approach for distributed systems.

1. Introduction

Software architecture is now widely recognized as a crucial aspect of software design that affects the entire software life-cycle. However, because software systems are seldom documented properly, and because design abstractions tend not to be explicitly represented in implemented software artifacts, reasoning about the architecture of an existing software system (being it for maintenance, reengineering or reuse) can be exceedingly difficult.

Architecture recovery aims at supporting the process of extracting up-to-date architectural information about an existing software system from its available software artifacts. Technically, architecture recovery can be seen as an attempt to reverse the process that might have been used to derive the existing software artifacts from a suitable architectural specification. However, any attempt to bridge the gap between an architecture specification and its implementation is hampered by the fact most traditional programming languages fail to provide high-level support for the implementation of design concepts[1]. In particular, those languages provide no explicit construct to implement distributed runtime abstractions (e.g., servers, clients, interaction protocols),

which end up encoded in more primitive programming language constructs such as data structures, functions, and classes.

Another difficulty stems from the fact that a single architectural description is generally insufficient to represent the design of a large distributed system. Industry practice has shown that a more effective approach is to describe a distributed system architecture from multiple perspectives or views, with each view emphasizing a specific set of concerns and including its own types of design elements and rationale[2][3]. As a consequence, architecture recovery should be concerned not only with the ability to recognize abstractions from different views, but also with the ability to understand how the abstractions of each view may relate to those of the other views[4].

2. Limitations of Current Approaches

Although there is considerable body of work on reverse engineering techniques and tools, these are generally limited in supporting architecture recovery because they tend to neglect aspects of system distribution and runtime organization. The exception are dynamic analysis tools (e.g., [5]) and domain-based pattern matching tools (e.g., [6][7]).

Dynamic analysis tools, though precise in capturing runtime elements and interactions, are generally limited in revealing how those elements may relate to, and are realized in terms of, the various elements of the source code. In addition, those tools tend to require costly event-tracing techniques, such as code instrumentation and symbolic execution, which may be impractical for time-critical systems or systems whose original development environment is no longer (or only partially) available.

Domain-based pattern matching tools in turn trade off precision against a better mapping of potential runtime events to their executing code fragments, but are still limited in revealing how those fragments may be reused across the code for different executable components.

3. Towards an Integrated Approach

We claim that integrating traditional and more sophisticated static analysis techniques can be a cost-effective way of recovering a distributed system's "as-built" architectural design. To support this claim, we have developed X-ray, an integrated approach for statically recovering distributed system architectural views[8].

X-ray comprises three domain-based static analysis techniques, namely *component module classification*, *syntactic pattern matching*, and *structural reachability analysis*. These complementary techniques can facilitate the task of identifying a distributed system's implemented executable components and their potential runtime interconnections. The component module classification technique automatically distinguishes source code modules according to the executables components they implement. The syntactic pattern matching technique in turn helps to recognize specific code fragments that may implement typical component interaction features. Finally, the structural reachability analysis technique aids in the association of those features to the code specific for each executable component.

By considering different types of domain information at multiple levels of abstraction, X-ray also helps to relate implemented design elements across multiple architectural views. For example, following Kruchten's 4+1 model[2], the module classification technique maps elements of the development view (source code modules and subsystems) to elements of the process view (executable components). The pattern matching and reachability analysis techniques further enrich this mapping by associating a module's specific code fragments to potential runtime interaction events. These two techniques are also useful in revealing aspects of the logical and physical views. For instance, recognizing that two executable components may interact at runtime through a pipe indicates that these components are likely to play the architectural role of filters ("logical view"), and that they can only be executed concurrently under the same communications domain ("physical view").

A preliminary investigation on the effectiveness of X-ray has been carried out through a number of case studies involving distributed software of varying sizes and application domains[9]. In particular, the approach has been successfully used to help extract architectural runtime information from the source code for two moderate-size, publicly-available distributed systems, namely Samba, a well-known file and print sharing service software suit, and Field, a distributed programming environment developed at Brown University. These experiments build confidence that combining module classification, syntactic pattern matching, and structural reachability analysis can be a cost-effective way of

recovering up-to-date architectural information from existing distributed system artifacts.

We are currently applying X-ray to assess distributed system evolution from a runtime architecture perspective. We also plant to investigate how the approach can be extended so as to support extraction of distributed runtime abstractions involving both physical and logical mobility[10].

References

- [1] M. Shaw and D. Garlan, "Software architecture: Perspectives on an emerging discipline", Prentice-Hall, 1996.
- [2] P. B. Kruchten, "The 4+1 view model of architecture", *IEEE Software*, 12(6):42-50, 1995.
- [3] C. Hofmeister *et al.*, "Applied software architecture", Addison Wesley, 2000.
- [4] B. Waters and G. Abowd, "Architectural synthesis: Integrating multiple architectural perspectives", In *WCRE'99*, IEEE CS Press, 1999.
- [5] D. Jerding and S. Rugaber, "Extraction of architectural connections from event traces", In *PASTE'98*, ACM Press, 1998.
- [6] R. Fiutem *et al.*, "A cliché based environment to support architectural reverse engineering", In *ICSM'96*, IEEE CS Press, 1996, pp. 319-328.
- [7] L. J. Holtzblatt *et al.*, "Design recovery for distributed systems", *IEEE Trans. on Softw. Eng.*, 23(7):461-471, 1997.
- [8] N. C. Mendonça and J. Kramer, "An Approach for Recovering Distributed System Architectures", *Automated Software Engineering Journal*, 8(3/4):311-354, August 2001.
- [9] N. C. Mendonça, "Software architecture recovery for distributed systems", PhD Thesis, Imperial College, Department of Computing, 1999.
- [10] G.-C. Roman, G. P. Picco, and A. Murphy, "Software engineering for mobility: A road map", In A. Finkelstein, editor, *The Future of Software Engineering*, ACM Press, June 2000, pp. 241-258.