AA – A Software Architecture Aware Environment for Dependable Systems

Cristina Gacek
School of Computing Science
Newcastle University — UK
cristina.gacek@ncl.ac.uk

Abstract

Explicitly considering software architectural information at all times is now a recognized means for addressing software system dependability. In this paper we propose the basic ideas for AA, an architecture aware environment to improve software system dependability. It builds on ideas from architecting dependable systems, control engineering, and software product lines. AA supports fault tolerance to also take into account global software architectural issues rather than only localized information (or immediate propagation), as well as viable variations in the software architecture.

1. Introduction

The software architecture of a software system provides an abstraction of its structure. Architectural focus has traditionally been on rigorous design, while dependability focus has traditionally been on lower levels of abstraction and the acceptance that residual faults will always be present. Architecting dependable systems focuses on reasoning about dependability at the architectural level [1]. This implies reflecting dependability considerations while architecting, as well as considering architectural information for verification and validation, fault tolerance, and system evaluation.

Current approaches to fault tolerance that consider architectural information focus on structuring guidelines and on providing reconfiguration support. Structuring guidelines aim at error containment, while reconfiguration aims at adapting a system in the context of the architectural elements directly impacted by the fault. These approaches alone may not suffice to provide meaningful and all encompassing fault tolerance in the true architecting dependable systems sense. This is so because not all architectural dependencies arise from localized interactions. Only some subsets of all possible architectural combinations are viable configurations of a software system. Architectural dependencies may exist because of communication effects or reliance on services provided from directly linked architectural elements. However, architectural dependencies may also exist among architectural elements that are not adjacent to each other. These dependencies can take several forms, including logical dependencies from the problem domain (e.g. on a financial system monitoring the stock market it makes no sense to generate reports on observed peaks in specific stock quotes information from a database if the quotes information is coming from an unreliable source) and physical dependencies from the deployment environment (e.g. using component X to monitor the vital signs of a patient consumes so many system resources that it is no longer viable for the system to also control the dosage of medication being delivered to the patient).

In this paper we argue that it is possible to address architectural dependencies while providing fault tolerance in a systematic fashion, rather than on a system specific one. We illustrate how this could be achieved by proposing the basic framework for AA, an architecture aware environment to improve software system dependability. In section 2 we briefly introduce concepts used to inspire this work. This is followed by a short description of the AA environment, and some brief conclusions.

2. Background

The software architecture of a software system provides an abstraction of its structure. It is usually described in terms of its components, connectors and their configuration [2; 3]. The way a software architecture is configured defines how various connectors are used to mediate the interactions among components.

The software product lines community has clearly identified that not all possible combinations of architectural elements are viable or even meaningful options for an individual system built from pre-existing independent parts. They tackle this issue by using product line architectures, possibly supported by domain models [4]. Clarifying how the architectural parts may be combined requires making explicit the *dependencies* among them (e.g., elements may exclude one another or one element may make the integration of a second one a necessity). Additionally, in product line architectures there are three different kinds of variations that are likely to occur (called *variabilities*). These are: a single option, representing an element that may but does not have to be a part of the system; an alternative, requiring the choice of 1 in N elements to be integrated; and a multiple choice, representing a mandatory architectural part consisting of multiple optional architectural elements. These abstractions are just as relevant for providing fault tolerance while architecting dependable systems, as the same constraints can be observed when adapting any software architecture.

In control theory it has long been recognized that there are many systems where the sole focus is on monitoring certain variables, which in turn may indicate that the system needs some adjustment such that the output of the system maintains some pre-specified properties. This type of system has inspired the definition of the control loop architectural style.

3. The AA Environment

The AA environment exhibits the control loop architectural style. It relies on monitoring systems at run time to trigger local and/or global system architectural adaptation. AA embraces the adoption of known fault tolerance techniques and approaches, while augmenting them with explicit software architecture information, including dependencies and variabilities.

AA requires, at run time: the adoption of fault tolerance techniques to immediately provide local architectural adaptation in response to exceptions raised; that all exceptions raised be logged; the existence of a system level monitor tracking the raised exceptions and local adaptations; architectural models including dependencies and variabilities; and a representation of the software architecture that is being used by the system.

The system level monitor aims at ensuring that the most appropriate architectural configuration is being used at all times. It is triggered when some local architectural adaptation takes place in response to some exception. It then accesses the exception log for an indication of which architectural elements are directly impacted and how (e.g. restarted, removed, replaced, or added). With this information at hand along with the explicit architectural constraints in the form of dependencies and variabilities, an adaptation strategy is pursued. This strategy enforces the known architectural constraints, while adopting the least intrusive adaptation option. System level monitors can be duplicated, and they can fail without drastically affecting the system's usual operation (crash or omission failures only). The observed impact would be that of having localized architectural adaptation only.

The architectural models may change over time. This can be done while the system is running, but it is unclear if it could be done while the system monitor is running.

The logged information on the observed facts surrounding an exception can be analyzed for common patterns of problematic conditions. This is done by observing which were the components and connectors directly involved, what other architectural elements were present, the resulting architectural changes, and the observed system stability after the adaptation. This would enable the detection of elements that never get used or that fail frequently, as well as problem combinations of architectural elements resulting from unforeseen emergent characteristics. This analysis of run time trends can better inform the directions that software system maintenance and evolution should take.

4. Conclusions and Open Issues

The AA environment proposed here has the potential to improve software system dependability by addressing issues of architecting dependable systems both at design time, by enticing the mitigation of architectural constraints to be considered when supporting fault tolerance, and at run time, by providing fault tolerance informed by various architectural constraints. AA also provides means to support architecturally informed system maintenance and evolution by observing run time trends.

AA is at an early stage of development. It requires further development of the underlying ideas and their validation, as well as further exploration of issues relating to the usage of AA in practice.

5. References

[1] Architecting Dependable Systems, http://www.cs.kent.ac.uk/people/staff/rdl/ADSFuture/index.htm, last accessed 28th March 2008.

[2] D.E. Perry and A.L. Wolf, "Foundations for the Study of Software Architectures", *SIGSOFT Software Engineering Notes*, vol. 17, no. 4, ACM, October 1992, pp. 40-52.

[3] Shaw, M. and D. Garlan, *Software Architectures: Perspectives on an Emerging Discipline*, Prentice-Hall, Inc., 1996.

[4] Bayer, J., O. Flege, P. Knauber, R. Laqua, D. Muthig, K., Schmid, T. Widen, and J. DeBaud, "PuLSE: a methodology to develop software product lines", *Proc. of the Symposium on Software Reusability*, ACM, May 1999, pp. 122-131.