Architecting Fault Tolerance using Abstractions

Patrick H. S. Brito[1*] Rogério de Lemos[2]
Cecília M. F. Rubira[1]

[1] State University of Campinas, Brazil
{pbrito, cmrubira}@ic.unicamp.br

[2] University of Kent, UK
r.delemos@kent.ac.uk

Abstract

This paper discusses how architectural abstractions can be effective in developing fault-tolerant software systems. Depending on the fault model and the resources available, different abstractions can be employed in order to represent explicitly issues that are related to fault tolerance, such as, error detection, and error and fault handling. These architectural abstractions, and their internal views, can be instantiated into concrete components and connectors for designing fault-tolerant software architectures. Since structural and behavioural properties associated with these abstractions are formally specified, the process of verifying and validating software architectures can be automated.

1. Introduction

The provision of fault tolerance relies on the existence of redundancy, which can be incorporated either implicitly or explicitly at the architectural level. An example of implicit redundancy is the usage of exception handling for supporting error recovery. If special care is not taken when structuring the system, the normal and abnormal specifications can be entangled thus increasing system complexity. Explicit redundancy is an inherent aspect of strongly-structured systems, i.e., systems in which the structuring of redundancy is part of the actual system, thus restricting the impact of faults. Examples of explicit redundancy are N-version programming and recovery blocks, which are two software fault tolerance techniques.

This paper presents how architectural abstractions can be effective when developing fault-tolerant software systems, applying both implicit and explicit redundancy. The idealised fault-tolerant architectural element (iFTE) [2], which is an abstraction based on exception handling, provides the means for promoting error confinement and supporting fault tolerance at the architectural level. The halt-on-failure architectural element (HoFE), which is an abstraction that assumes crash failure semantics, provides the basis for incorporating explicit redundancy when designing fault-tolerant systems. Depending on the fault model and the availability of resources, the appropriate architectural abstraction should be used. For obtaining fault-tolerant software architectures, these abstractions are instantiated into architectural components and connectors, which are then configured depending on the interaction constraints dictated by their structural and behavioural properties.

These architectural abstractions are presented in the context of a general approach for the formal specification, verification, and validation of fault-tolerant systems. The adoption of abstractions together with the use of formal languages allows the automatic verification of high-level models for identifying and removing design faults at the initial stages of the software lifecycle. After the verification activities, the architectural models can be used as a basis for generating both the system's source code [3], and its architectural-based test cases [4]. The automatic transformation from architectural models into source code tends to reduce the number of faults that are introduced into the system implementation when compared with an error prone manual programming. The architectural-based test cases are able to identify and remove implementation faults that are related to the architectural design of the system, although they are restrictive on their system test coverage.

2 Fault-tolerant Architectural Abstractions

The *idealised fault-tolerant architectural element* (iFTE) is an architectural abstraction for structuring fault-tolerant systems. This abstraction enforces the principles associated with the concept of the idealised fault-tolerant component [1], and incorporates mechanisms for detecting errors, as well as propagating and handling them in a structured way. The iFTE abstraction provides an explicit separation of concerns between two types of behaviour: (i) the nor-

*Supported by Fapesp/Brazil, grant 06/02116-2 and CAPES/Brazil, grant 0722-07-3.

mal behaviour, which realises the services of the application, and (ii) the abnormal (exceptional) behaviour, which realises the detection, propagation and handling of errors. In order to provide this separation, the iFTE abstraction defines four types of interfaces, which are presented in Figure 1. While the I_iFTE_PN and I_iFTE_RN are responsible for the normal behaviour, I_iFTE_PA and I_iFTE_RA are responsible for the abnormal behaviour.

Figure 1. iFTE Abstraction

The *halt-on-failure architectural element* (HoFE) is an architectural abstraction for the provision of error confinement and fault tolerance, and which enforces the principles associated with the *crash failures* fault model [5]. When an HoFE fails, it fails silently without producing any error signal. The HoFE abstraction defines two types of interfaces, which are presented in Figure 2. It is assumed that an HoFE is able to detect failures on other architectural elements from which requests operations, e.g., by associating *time-outs* with the I_HoFE_Req interfaces.

Figure 2. HoFE Abstraction

3 A Rigorous Development Method

In our approach, abstractions are first-level units, guiding the development since the specification, until the verification and validation of the software architecture. An overview of the development method is shown in Figure 3.

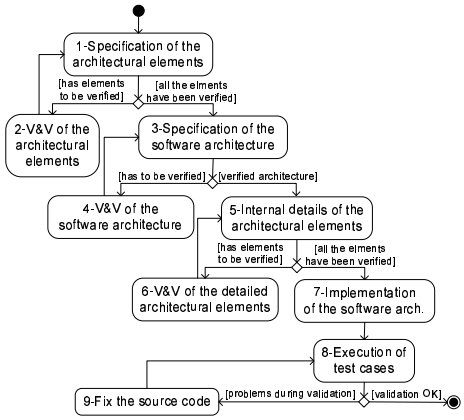

Figure 3. A Rigorous Development Method

Figure 4 details the execution of Activities 2, 4, and 6 of Figure 3, which refer to the verification and validation (V&V) of the software architecture. A main feature of our proposed approach is that the scenario concept is used in all the V&V activities.

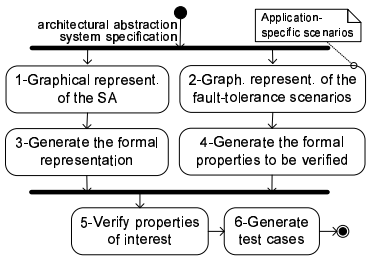

Figure 4. Steps of the V&V Activities

4 Conclusions

The development of fault-tolerant software system can be more effective if architectural abstractions are employed. These are able to abstract away from system details while providing the means for analysing how errors are propagated, detected and handled, and how faults are handled. Associated with these abstractions, we have defined a general rigorous development approach for the formal specification, verification and validation of software architectures that are based on these abstractions. In this paper, we have presented two distinct architectural abstractions from which fault-tolerant software systems can be built: the idealised fault-tolerant architectural element (iFTE), and the halt-on-failure architectural element (HoFE).

References

[1] T. Anderson and P. A. Lee. *Fault Tolerance: Principles and Practice*. Prentice-Hall, 1981.

[2] R. de Lemos. Architectural fault tolerance using exception handling. In R. de Lemos, C. Gacek, and A. Romanovsky, editors, *Architecting Dependable Systems IV, LNCS 4615*, pages 142–162, 2007.

[3] S. Entwisle, H. Schmidt, I. Peake, and E. Kendall. A model driven exception management framework for developing reliable software systems. In *Proc. of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 307–318, Washington, DC, USA, 2006. IEEE Computer Society.

[4] D. J. Richardson and A. L. Wolf. Software testing at the architectural level. In *International Workshop on Multiple Perspectives in Software Development (Viewpoints '96) on SIGSOFT '96 Workshops*, pages 68–71, New York, NY, USA, 1996. ACM.

[5] G. Varghese and M. Jayaram. The fault span of crash failures. *Journal of the ACM*, 47(2):244–293, 2000.