INTRODUCTION AND OBJECTIVES

The programming language itself is the focus of this chapter: Fault-tolerance is not embedded in the program (as it is the case e.g. for single-version fault-tolerance), nor around the language (through compilers or translators); on the contrary, fault-tolerance is provided through the syntactical structures and the run-time executives of fault-tolerance programming languages. Also in this case a significant part of the complexity of dependability enforcement is moved from each single code to the architecture, in this case the programming language.

Many cases exist of fault-tolerance programming languages; this chapter proposes a few of them, considering three cases: Object-oriented languages, functional languages, and hybrid languages. In particular it is discussed the case of Oz, a multi-paradigm programming language that achieves both transparent distribution and translucent failure handling.

FAULT-TOLERANT PROTOCOLS USING CUSTOM PROGRAMMING LANGUAGES

Another approach is given by working at language level enhancing a pre-existing programming language or developing an ad hoc distributed programming language.
so that it hosts specific fault-tolerance provisions. The following two sections cover these topics.

**Object-Oriented Approaches**

**The Arjuna Distributed Programming System**

Arjuna (Arjuna Technologies, ltd.) is an object-oriented system for portable distributed programming in C++ (Shrivastava, 1995). It can be considered as a clever blending of useful and widespread tools, techniques, and ideas—as such, it is a good example of the evolutionary approach towards application-level software fault-tolerance.

Arjuna exploits remote procedure calls (Birrell & Nelson, 1984) and UNIX daemons (Haviland & Salama, 1987). On each node of the system an object server connects client objects to objects supplying services. The object server also takes care of spawning objects when they are not yet running (in this case they are referred to as “passive objects”). Arjuna also exploits a “naming service”, by means of which client objects request a service “by name”. This transparency effectively supports object migration and replication. Arjuna offers the programmer means for dealing with atomic actions (via the two-phase commit protocol) and persistent objects. Unfortunately, it requires the programmers to explicitly deal with tools to save and restore the state, to manage locks, and to declare in their applications instances of the class for managing atomic actions. As its authors state, in many respects Arjuna asks the programmer to be aware of several complexities—as such, it is prejudicial to transparency and separation of design concerns (insufficient sc). On the other hand, its good design choices result in an effective, portable environment.

Arjuna provides the programmer with “a computation model in which application programs manipulate persistent objects under the control of atomic actions” (Shrivastava, 1995). This concise definition reflects the opinions of many fault-tolerance language designers, that is, that coupling the object and atomic action model with object persistency provides a good solution to designing fault-tolerant systems. In Arjuna a failure appears to the programmer as a persistent object becoming unavailable. Actual reasons for this event may be e.g. a crash of the object repository, or a network partitioning. Accordingly, avoiding failures in Arjuna is achieved by any method directed to increasing the availability of its objects. The main approach used for this in Arjuna is object replication. Like in any replication scheme, clearly this calls for consistency protocols among the replicas. As mentioned above, Arjuna uses an object server to turn passive objects into active ones, when a reference to any such object is detected in the system. Binding is the name of the operation that
Related Content

Exploration and Exploitation of Developers' Sentimental Variations in Software Engineering
[www.igi-global.com/article/exploration-and-exploitation-of-developers-sentimental-variations-in-software-engineering/166542?camid=4v1a](www.igi-global.com/article/exploration-and-exploitation-of-developers-sentimental-variations-in-software-engineering/166542?camid=4v1a)

A Comparative Analysis of Software Engineering Approaches for Sequence Analysis
[www.igi-global.com/chapter/comparative-analysis-software-engineering-approaches/75751?camid=4v1a](www.igi-global.com/chapter/comparative-analysis-software-engineering-approaches/75751?camid=4v1a)
A Comparative Analysis of Reliability Assessment Methods for Web-Based Software
Jinhee Park, Yeong-Seok Seo and Jongmoon Baik (2013). International Journal of Software Innovation (pp. 31-44).
www.igi-global.com/article/a-comparative-analysis-of-reliability-assessment-methods-for-web-based-software/105630?camid=4v1a

Automating Web Service Composition: An Ontological Agent Framework
www.igi-global.com/chapter/automating-web-service-composition/115434?camid=4v1a