INTRODUCTION AND OBJECTIVES

Failure detection is a fundamental building block to develop fault-tolerant distributed systems. Accurate failure detection in asynchronous systems (Chapter II) is notoriously difficult, as it is impossible to tell whether a process has actually failed or it is just slow. Because of this, several impossibility results have been derived—see for instance the well-known paper (Fischer, Lynch, & Paterson, 1985). As a consequence of these pessimistic results, many researchers have devoted their time and abilities to understanding how to reformulate the concept of system model in a fine-grained alternative way. Their goal was being able to tackle problems such as distributed consensus with the minimal requirements on the system environment. This brought to the theory of unreliable failure detectors for reliable systems, pioneered by the works of Chandra and Toueg (Chandra & Toueg, 1996). This chapter introduces these concepts and the formulation of failure detection protocols in the application layer. In particular a linguistic framework is proposed for the expression of those protocols. As a case study it is described the algorithm for failure detection used in the EFTOS DIR net and in the TIRAN Backbone—that is, the fault-tolerance managers introduced respectively in Chapter III and Chapter VI.
FAILU RE DETECTION PROTOCOLS IN THE APPLICATION LAYER

In Chapter II the concept of system model was briefly introduced together with the main features of the classical asynchronous and synchronous system models. The former model, also known as “time-free” system model, is the one that is implicitly used by most non real-time services: For such systems there is no bound for the time required to execute any computation or communication step, which means that there is no way to tell whether a certain part of the system is slow or if it has failed. How to distinguish between these two cases? The answer found by researchers is failure detectors. As cleverly expressed by Michel Raynal, failure detectors may be considered as a sort of distributed oracle for failure detection. This oracle observes the system and draws its conclusions about failures, informing those who query it. It could be regarded as a sort of middleware service for failure detection. Failure detectors are characterized by two properties:

• Completeness, which is the actual detection of failures, and
• Accuracy, which tells how reliable a failure detector can be in its assessments.

In a sense, completeness and accuracy represent two coordinates by means of which the spectrum of all possible failure detectors can be drawn. This spectrum represents also a two-dimensional set of possible system models, a set which is much more detailed and fine-grained than the linear one hitherto available to researchers:

In other words, before failure detectors, the researchers had a sort of interval defined by its two extremes, the asynchronous model (“I ask nothing, so I get nothing”) and the synchronous model (“I ask too much, so I can’t get it”). Partial synchronous systems are points vaguely identified within that interval. With failure detectors everything changes and one can talk of system model \((c, a)\), where \((c, a)\) are the completeness and accuracy of the minimal failure detector \(FD(c,a)\) that can be implemented in a system obeying that model. This view has revolutionized the research on dependable distributed systems.

Several and very important have been the consequences of the introduction of failure detectors. Among them the following ones are highlighted herein:

• Famous impossibility results such as the impossibility to solve consensus in a fully asynchronous system (Fischer et al., 1985), which had puzzled the
Related Content

**Formalization of Expert Knowledge About the Usability of Web Pages Based on User Criteria Aggregation**

**Evaluating Performance of Software Architecture Models with the Palladio Component Model**
[www.igi-global.com/chapter/evaluating-performance-software-architecture-models/26827?camid=4v1a](www.igi-global.com/chapter/evaluating-performance-software-architecture-models/26827?camid=4v1a)
Assuring Maintainability in Model-Driven Development of Embedded Systems
www.igi-global.com/chapter/assuring-maintainability-model-driven-development/26836?camid=4v1a

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