INTRODUCTION AND OBJECTIVES

In this chapter our survey of methods and structures for application-level fault-tolerance continues, getting closer to the programming language: Indeed, tools such as compilers and translators work at the level of the language—they parse, interpret, compile or transform our programs, so they are interesting candidates for managing dependability aspects in the application layer. An important property of this family of methods is the fact that fault-tolerance complexity is extracted from the program and turned into architectural complexity in the compiler or the translator.

Apart from continuing with our survey, this chapter also aims at providing the reader with two practical examples:

- Reflective and refractive variables, that is, a syntactical structure to express adaptive feedback loops in the application layer. This is useful to resilient computing because a feedback loop can attach error recovery strategies to error detection events.
- Redundant variables, that is, a tool that allows designers to make use of adaptively redundant data structures with commodity programming languages such as C or Java. Designers using such tools can define redundant data structures...
in which the degree of redundancy is not fixed once and for all at design time, but rather it changes dynamically with respect to the disturbances experienced during the run time.

Both tools are new research activities that are currently being carried out by the author of this book at the PATS research group of the University of Antwerp. It is shown how through a simple translation approach it is possible to provide sophisticated features such as adaptive fault-tolerance to programs written in any language, even plain old C.

FAULT-TOLERANT PROTOCOLS USING COMPILERS AND TRANSLOGRS

Our first subject is tools that work “within” the compiler: Meta-object protocols. Most of such tools are based on the concept of reflection: The ability to mirror the feature of a system by creating a causal connection between sub-systems and internal objects. In other words, events experienced by a reflected sub-system trigger events on the object representing that sub-system, and vice-versa.

Compiler-Level Tools: Meta-Object Protocols, Reflection, and Introspection

Some of the negative aspects pointed out while describing single and multiple version software approaches can be in some cases weakened, if not solved, by means of a generic structuring technique that allows in some cases an adequate degree of flexibility, transparency, and separation of design concerns to be reached: the adoption of meta-object protocols (Kiczales, Rivières, & Bobrow, 1991). The idea is to “open” the implementation of the run-time executive of an object-oriented language like C++ or Java so that the developer can adopt and program different, custom semantics, adjusting the language to the needs of the user and to the requirements of the environment. Using meta-object protocols, the programmer can modify the behavior of a programming language’s features such as methods invocation, object creation and destruction, and member access. The transparent management of spatial and temporal redundancy (Taylor, Morgan, & Black, 1980) is a context where meta-object protocols appear to be particularly adequate.

The key concept behind meta-object protocols is that of computational reflection, or the causal connection between a system and a meta-level description representing structural and computational aspects of that system (Maes, 1987). The protocols offer the meta-level programmer a representation of a system as a set of meta-objects, i.e., objects that represent and reflect properties of “real” objects, i.e., those
High-Integrity Model-Based Development
www.igi-global.com/chapter/high-integrity-model-based-development/117937?camid=4v1a