Abstract

Hilderink, G.H., *"Managing Complexity of Control Software through Concurrency"*, Public Defence on May 19, 2005, University of Twente, pp ix - 352, ISBN 90-365-2204-8, 2005 (Supervised by Amerongen, J. van and J.F. Broenink)

In this thesis, we are concerned with the development of concurrent software for embedded systems. The emphasis is on the development of control software.
Embedded systems are concurrent systems whereby hardware and software communicate with the concurrent world. Concurrency is essential, which cannot be ignored. It requires a proper handling to avoid pathological problems (e.g. deadlock and livelock) and performance penalties (e.g. starvation and priority conflicts). Multithreading, as such, leads to sources of complexity in concurrent software. This complexity is considered frightening, because it complicates the software designs and the resulting code. Moreover, this paradigm complicates the understanding of the behaviour of concurrent software. A paradigm with a precise understanding of concurrency is essential. In this thesis, a methodology is proposed that comprises a paradigm of fundamental aspects of concurrency. These fundamental aspects are derived from the Communicating Sequential Processes (CSP) theory. CSP is a theory of programming that is developed by Hoare, Brookes, and Roscoe. CSP comprises fundamental concepts useful for precisely describing and studying concurrent systems. These concepts are based on processes and events. Processes and events are abstract entities, more abstract than objects. Processes and events are essential in describing and reasoning about the real-time behaviour of process architectures. A process architecture describes a (sub-) program as a composition of communicating processes.
The proposed methodology brings a subset of CSP to practice in order to specify, design, and implement process architectures. The CSP concepts bring forth a glue-logic between these phases in the development trajectory. Furthermore, these concepts offer technical solutions, which have been enhanced with notion of priorities, exception handling, and timing. The precise semantics of the concepts and their restrictions provide the guidelines to create reliable and robust concurrent software. The abstraction and separation of well-defined concerns contribute to managing complexity in concurrent software.
The proposed methodology defines the following ingredients:
1. A graphical modelling language defines graphical notations and rules that are derived from CSP. The graphical modelling language is used for specifying, designing, and graphically programming process architectures. This results in CSP diagrams.
2. An object model implements the CSP concepts by means of objectoriented techniques. This model can be implemented in objectoriented programming languages. This results in the CSP libraries for Java, C (in object-oriented style) and C++.
The graphical modelling language and the object model go together. CSP diagrams are used to describe and to analyse process architectures. A CSP library is used to implement process architectures in an objectoriented programming language. This methodology uses processoriented and object-oriented techniques, and hides thread-oriented techniques.
The proposed methodology is applied to control applications on embedded computer systems.