Abstract. The development of concurrent applications, which consist of several parallel, often distributed, activities that communicate and synchronize with each other, presents challenging logical and technological issues.

In the common practice, technological aspects, which are related to the definition of concurrent processes as well as of their interactions, often place constraints on the logical ones. Consequently, the flexibility that could be attained during design is reduced and, further, an actual implementation where both aspects are deeply intermixed is usually obtained. This paper first presents an expressive and flexible language based on object-oriented high-level nets, which allows designers to naturally represent concurrency and synchronization. Then, it illustrates a powerful software engineering environment, CAB (Concurrent Applications Builder), which supports the building and the simulation/animation of models as well as the automatic generation of applications from models (by transforming the model's objects into implementation processes). A case study from an actual project is illustrated.

1 Introduction

The complexity of today's systems, such as telecommunications systems, real-time embedded systems and automated manufacturing control systems, calls for the development of an increasing number of concurrent applications.

A concurrent application (CA) consists of a group of activities which execute in parallel and communicate and synchronize with each other. Parallelism is real when there are several computing units; it is logical when several activities which are logically parallel share a single computing unit.

The difficulties that characterize the development of a CA can be subdivided into two categories:

1. logical, since it is necessary to effectively express concurrent activities as well as their interactions;
2. technological, because low-level implementation mechanisms related to the underlying operating system (OS) and network facilities must be mastered properly.
In the common approach, technological aspects exert great influence on logical ones (e.g., the one-to-one correspondence existing between design entities and implementation processes) reducing, on the one hand, the flexibility that can be attained during the early phases of analysis and design and, on the other hand, leading to an actual implementation where both aspects are deeply intermixed.

In this paper, after a detailed analysis of the common approach to the development of CAs, two directions of improvement are examined, one based on a more expressive design language (Protob), the other on an innovative software engineering environment (CAB).

The language, Protob, is based on object-oriented high-level nets and allows designers to naturally represent concurrency and synchronization. The software engineering environment, CAB (Concurrent Applications Builder), supports the building and the simulation/animation of models as well as the automatic generation of applications from models (by transforming the model's objects into implementation processes). The last feature, i.e., the automatic generation of processes from Protob objects, is emphasized in this paper.

A case study from an actual project is illustrated and the proposed approach is compared with the traditional one.

2 The traditional approach and its drawbacks

In order to highlight the typical problems encountered in the development of a CA, we refer to the major phases, i.e., analysis, design and implementation, of a waterfall life cycle. However, the considerations made below can apply to other life cycles as well.

2.1 Analysis

In this phase, a model, called a specification model, of the CA is built according to extended functional approaches, such as SA/RT [1, 2], or object-oriented ones such as OO/CAD [3]. The specification model is basically a hierarchical structure consisting of logically concurrent units which will be referred to as analysis concurrent units (ACUs).

There are two major kinds of ACUs: state-based ACUs, which are usually modeled by state-transition diagrams (STDs), and functional ACUs, which are based on the functional decomposition. Emphasis is placed on the logical decomposition of the CA into ACUs, while architectural issues as well as interaction mechanisms are generally put off until design.

2.2 Design

The aim of this phase is to transform the specification model into a design model in such a way that it will then be straightforward to code it and give rise to the actual CA.