Modelling Remote Concurrency with Ada
Case Study of Symmetric Non-deterministic Rendezvous

Claude Kaiser, Christophe Pajault, and Jean-François Pradat-Peyre

CEDRIC - CNAM Paris
292, rue St Martin, F-75003 Paris
{kaiser,christophe.pajault,peyre}@cnam.fr
http://quasar.cnam.fr/

Abstract. When developing concurrent software, a proper engineering practice is to choose a good level of abstraction for expressing concurrency control. Ideally, this level should provide platform-independent abstractions but, as the platform concurrency behaviour cannot be ignored, this abstraction level must also be able to cope with it and exhibit the influence of different possible behaviours. We state that the Ada language provides such a convenient abstraction level for concurrency description and evaluation, including distributed concurrency. For demonstrating it, we present two new cooperative algorithms based on remote procedure calls which, although simply stated, contain actual concurrency complexity and difficulties. They allow a distributed symmetric non-deterministic rendezvous. One relies on a common server and the second is fully distributed. Both realize a symmetric rendezvous using an asymmetric RPC modelled by Ada rendezvous. These case studies show that Ada concurrency features provide the adequate abstraction level both for describing and evaluating concurrency and for carrying out design decisions.

1 Introduction

1.1 The Need of High-Level Concurrency Description

Concurrency is a prolific source of complexity and is a serious cause of errors when developing software. Thus it is a challenge for developers of long-lived, high-quality software that needs reliable software technologies.

Current approaches to software development use patterns or models as a set of guidelines for structuring application specification, design and implementation. Providing significant examples is of prime importance for mastering the additional temporal dimensions of correctness introduced by concurrency, i.e., safety and liveness. Even if you never employ them directly, reading about different special-purpose design patterns can give you ideas about how to attack real problems.

Moreover when developing concurrent software, a proper engineering practice is to choose a good level of abstraction for expressing concurrency control. Ideally, this level should provide platform-independent abstractions. However the
concurrency semantics of platforms associated with POSIX standards or with languages like Ada, Java or C# are different and this diversity may influence the correctness of some models or patterns. In [7], we have shown examples where the weak liveness semantics of Java and C# run time causes deadlock in some programs, which nevertheless have been proven safe with the Ada strong liveness semantics. As the platform concurrency behaviour cannot be ignored, the abstraction level should be able to cope with it and to analyse the influence of the different possible behaviours.

1.2 Ada as a Concurrency Description and Modelling Language

We state that the Ada language provides such a convenient abstraction level and thus may be used for concurrency description and evaluation, including distributed concurrency. Our statement is based on four assumptions.

First Ada proposes today the most powerful set of high level concurrency features available in an imperative language and its concurrency semantics is well and precisely defined. For expressing cooperation through a shared memory, protected objects can be used together with the requeue statement and with the entry family facility. For analysing communication without a shared memory, the rendezvous together with the use of the requeue statement allows to simulate simply the semantics of a remote procedure call.

Second the behavioural semantics of shared memory platforms used for other languages such as POSIX, C# or Java can be emulated with Ada [7]. Similarly, the remote procedure call, which is used in message passing protocols, can be simulated by Ada task rendezvous.

Third, as Ada concurrency semantics is precisely defined, model programs expressed in Ada can be analysed automatically for detecting correctness deficiencies such as deadlock or starvation. Our tool QUASAR [6], based on slicing, followed by Petri net generation and by model checking of the generated net, is devoted to concurrent Ada programs analysis. It allows evaluating and validating a concurrency model description at the design and specification stage.

Fourth, Ada provides an executable description language. This allows running simulations and testing the concurrency behaviour of programs.

Indeed, our approach that we teach also to our students aims at mixing design and evaluation. We are convinced that this encourages choosing simpler concurrency architectures in order to render them more readable, understandable, and finally easier to validate, maintain, reuse and modify.

This is the most necessary, as our students are not lucid enough about concurrent programming; as many designers they underestimate its difficulties and the need of a language for coping with clear concurrency ideas and structures. Today’s programming approaches, whatever the pedantic name they use, are often close to cut-and-paste techniques and lack concurrency analysis.

We have already shown how data sharing paradigms which use the monitor concept [13] can be expressed in Ada and validated while running on platforms with different fairness semantics. We will now deal with remote procedure call.