Available online at www.sciencedirect.com
SCIENCE DIRECT®
Environmental Modelling & Software 20 (2005) 1081–1100
Environmental Modelling & Software
www.elsevier.com/locate/envsoft

Eclpss: a Java-based framework for parallel ecosystem simulation and modeling

Elaine Wenderholm

Computer Science Department, State University of New York at Oswego, Oswego, NY 13126 USA

Received 30 May 2003; received in revised form 10 May 2004; accepted 7 June 2004

Abstract

Eclpss (Ecological Component Library for Parallel Spatial Simulation) is a Java™-based framework designed to give ecologists the ability to easily develop grid-based ecosystem simulations at multiple spatial and temporal scales. The framework automatically targets the model to shared memory parallel machines. Because of the judicious use of Java, both the framework and framework-generated models are platform independent. Users may write arbitrarily complex models without the need to be expert programmers. These models are reusable, easily modifiable and extensible. Collaborative model development, sharing, and dissemination with automatically-generated documentation are all web-accessible. The modelling environment consists of a suite of GUI-based tools which are designed to be intuitive to ecologists. Ecologists specify the model; the Eclpss compiler uses these specifications to generate code. Scientific unit measurements are incorporated into specifications and consistency checking is performed; substance consistency is supported. This paper presents the structure and features of the Eclpss framework, the migration of a Matlab model into this framework, and concludes with a discussion of ongoing and planned future work.

Keywords: Spatial simulation framework; Java; Shared-memory parallel; Platform independence; Units; XML

1. Introduction

As ecological modelling becomes increasingly more comprehensive and complex, it becomes more difficult to reuse some or all pieces of a model.

In the ecological domain, there is the desire to both parallelize and reuse the code from several disparate programs whose commonality might only be that they are written in the same programming language. Accomplishing this may be viewed, in part, as generating code (either manually or automatically) which "glues" together different programs and, optionally, mapping the program to a parallel architecture. This typically poses more technical difficulties than the Fortran "dusty deck" problem of the late 1980s[1] which parallelizes just one program. Since each ecological model is a different program, the initial code design can render it inflexible to incorporation into larger model(s). This also is difficult to automate since general code is hard to analyze.

Eclpss takes a different approach to model design and building: disparate models are not combined; new Eclpss models are developed; these Eclpss models may then be combined.

Eclpss models are component-based (He et al., 2002). An Eclpss Component contains pieces of user-written

[1] Parallelizing compilers analyze sequential code and, using various loop transformations, automatically restructure it into parallel implementations. (See, for example, Allen and Kennedy (1987) and Ruhl and Annaratone (1990)).

E-mail address: wender@cs.oswego.edu (E. Wenderholm).
URL: http://www.cs.oswego.edu/~wender.

1364-8152/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.envsoft.2004.06.006

code. The framework imposes no restriction on the complexity of the code itself, but only on the component interface. Components are independent and may not directly reference other components; interaction is indirect via updates to state variables. An Eclpss model may be viewed as a circuit: components are the "chips"; state variables are the "wires" that connect components. This design allows components (and the models which use them) to be freely shared, interoperable, and interchangeable. Eclpss supports top–down and bottom–up design, debugging, and experimentation. Ecologists can easily rearrange and experiment with model structure, grids, grid cell size and scale.

A programming variable in scientific programs not only has a storage type, but often has a unit of scientific measurement. Different ecological models may (correctly) use different measurement units, different measurement systems, or both, for the same programming variable in different parts of the program. Models with multi-ecological media are typical of the use of different measurement (and hence modelling) units: the density in air (of, say, a nitrogen compound) may be measured in kg/m^3; the same compound in soil surface in mg/cm^3. Unit (and data) conversion at the ecological interface is necessary. Poorly-defined programming interfaces, such as the Mars Climate Orbiter, have lead to the incorrect use of different measurement systems. Since compilers for programming languages only perform storage type-checking, measurement unit type and consistency verification often requires that checking be done by hand.

The manner in which Eclpss models are developed has many of the same characteristics as specialized programming tools.

Specialized programming tools in other application areas (such as spreadsheets, relational database management systems and symbolic mathematics programs) have allowed a community of users to write applications that previously required specialist programmers and many person-months (person-years) of development and support time. Many users would be unable to develop such applications without the use of these specialized systems. In fact, these specialized tools are so widespread that they are taken for granted. They share several common characteristics:

- Each addresses a restricted and well-defined problem domain.
- The user interface is designed to be natural to the target user community.
- Features from declarative programming (viz., specifying "what" to compute instead of "how" to compute) are incorporated into the tools, thereby freeing the user from programming details.
- Some commonly support an automatic parallel implementation.
- Many increasingly are becoming web-based and/or web-accessible.

As a result, large communities of users may now develop fairly complex applications; most users would otherwise be unwilling or unable to develop such sophisticated applications.

These same characteristics are incorporated into the Eclpss framework:

- The modelling domain centers on grid-based simulations over time at multiple spatial scales. The calculation of each grid point depends on data within a smallish, localized neighborhood. In addition to the ecosystem modelling domain, other application areas include thermal diffusion, problems which give rise to PDEs, initial-value problems for ODEs, and cellular automata (El Yacoubi et al., 2003) with Cartesian neighborhoods.
- Models are specified using a suite of GUI-based tools which facilitate and support the design practices that are natural to ecologists.
- The ease of developing models is due largely to the declarative nature provided by the framework.

  Parts of a model are expressed at a high level of abstraction as *specifications*. Specifications relieve the user of the need to write (and rewrite) code that manages storage and other mundane but error-prone programming tasks such as the explicit declaration of data structures and loops; the Eclpss compiler uses the specifications to generate this code.
- The framework takes advantage of cutting-edge technology afforded by Java, which supports graphical, web-centric development, collaboration, and dissemination of models.
- Models are automatically targeted to the host machine.
- Most implementation details are invisible to the user.

As a consequence the user does not need a deep understanding of most of this generated code.

This understanding can be especially difficult for the code that is generated for parallel execution. This invisibility permits framework developers to add independent enhancements to both the framework code and the compiler.

In addition to simplicity for the user, the Eclpss compiler must generate efficient (parallel) code. The programming interface is simplified and restricted to framework `get` and `set` methods, which not only eases the conceptual task of the user, but also the analytical task of the Eclpss compiler. The framework rigorously enforces just enough structure, so that a model is relatively easy to analyze, but it does not impose too much structure, so that users have a high degree of modelling freedom.