The Concurrency Factory Software Development Environment

Rance Cleaveland (`rance@csc.ncsu.edu`)*
Philip M. Lewis (`pml@cs.sunysb.edu`)†
Scott A. Smolka (`sas@cs.sunysb.edu`)†
Oleg Sokolsky (`oleg@ccc.com`)†

ABSTRACT
The *Concurrency Factory* is an integrated toolset for specification, simulation, verification, and implementation of real-time concurrent systems such as communication protocols and process control systems. Two themes central to the project are the following: the use of *process algebra*, e.g., CCS, ACP, CSP, as the underlying formal model of computation, and the provision of *practical* support for process algebra. By "practical" we mean that the Factory should be usable by protocol engineers and software developers who are not necessarily familiar with formal verification, and it should be usable on problems of real-life scale, such as those found in the telecommunications industry.
This demo is intended to demonstrate the following features of the Concurrency Factory: the graphical user interface VTView and VTSim, the local model checker for the alternation-free modal mu-calculus, and the graphical compiler that transforms VTView specifications into executable code.

1 Introduction

The *Concurrency Factory* is an integrated toolset for specification, simulation, verification, and implementation of real-time concurrent systems such as communication protocols and process control systems. The project, which is a joint effort between SUNY Stony Brook and North Carolina State

*Department of Computer Science, N.C. State University, Raleigh, NC 27695-8206, USA. Research supported in part by NSF/DARPA grant CCR–9014775, NSF grant CCR–9120995, ONR Young Investigator Award N00014-92-J-1582, and NSF Young Investigator Award CCR–9257963, and AFOSR grant F49620-95-1-0508.

†Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794-4400, USA. Research supported in part by NSF grants CCR–9120995 and CCR–9208585, and AFOSR grants F49620-93-1-0250 and F49620-95-1-0508.

University, officially started in Spring '92, and is currently supported by grants from NSF and AFOSR. Two themes central to the project are the following: the use of *process algebra* (e.g., CCS, CSP, ACP) as the underlying formal model of computation, and the provision of *practical* support for process algebra. By "practical" we mean that the Factory should be usable by protocol engineers and software developers who are not necessarily familiar with formal verification, and it should be usable on problems of real-life scale, such as those found in the telecommunications industry.

The main features of the Concurrency Factory are the following:

- A *graphical user interface*, VTView/VTSim, that allows the non-expert to design and simulate concurrent systems using process algebra. VTView is a graphical editor for hierarchically structured networks of finite-state processes, and VTSim is a sophisticated environment for the simulation and testing of VTView-constructed specifications. We are currently extending the GUI to allow processes to be embedded in states of other processes, thereby permitting compact specifications such as those found in statecharts.

- A *textual user interface* for the language VPL, a simple language for concurrent processes that communicate values from a finite data domain. A VPL compiler translates VPL programs into networks of finite-state processes.

- A *suite of verification routines* that currently includes a linear-time, global model checker for L_{μ_1}, the alternation-free fragment of the modal mu-calculus [CS93], a local model checker for L_{μ_1} [Sok96], a local model checker for a real-time extension of L_{μ_1} [SS95], and strong and weak bisimulation checkers.

  Care is being taken to ensure that these algorithms are efficient enough to be used on real-life systems. For example, we are investigating how these algorithms can be parallelized [ZS92, lic94], and made to perform incrementally [SS94].

- A *graphical compiler* that transforms VTView and VPL specifications into executable code. Our current version produces Facile [GMP89] code, a concurrent language that symmetrically integrates many of the features of Standard ML and CCS. We are also considering adding a concurrent extension of C++ as another target language. The compiler relieves the user of the burden of manually recoding their designs in the target language of their final system.

The Concurrency Factory is written in C++ and executes under X-Windows, using Motif as the graphics engine, so that it is efficient, easily extendible, and highly portable. It is currently running on SUN SPARC-stations under SunOS Release 4.1. The basic organization of the system is depicted in Figure ??.

In what follows, we briefly describe VTView and VTSim, the main components of the demo, and a protocol verification case study.

2 Graphical Editor and Simulator

The graphical user interface of the Concurrency Factory consists of two main components, VTView and VTSim (VT stands for *Verifier's Toolkit*). VTView is a graphical tool supporting the design of hierarchically structured systems of communicating tasks expressed in GCCS, a graphical specification language. GCCS provides system builders with intuitive constructs (buses, ports, links, a subsystem facility, etc.) for designing systems, and it supports both top-down and bottom-up development methodologies. As user designs are entered using VTView, an internal representation is created by invoking the appropriate methods associated with process and network objects.

In contrast with other graphical languages, such as Harel's statecharts and Maraninchi's Argonaute, GCCS is designed to model systems in which processes execute asynchronously (although communication between processes is synchronous). The language is equipped with a formal semantics in the form of a structural operational semantics, à la Plotkin and Milner. The semantics has been "implemented" in the Factory in the form of methods that determine the set of transitions that are possible for a network or a process from a given state. Both the graphical simulator and the method that produces the global automaton from a network of process rely fundamentally on these methods. By encapsulating the semantics of VTView objects, all tools within the Factory are guaranteed to interpret processes and networks consistently.

VTSim permits users to simulate graphically the execution of GCCS systems built using VTView. The tool provides both interactive and automatic modes of operation, and it also includes features such as breakpoints and reverse execution. It also enables users to view the simulated execution of a system at different levels in the structure; one can either choose to observe the simulation at the interprocess level and watch the flow of messages, or one can look at individual processes in order to see why messages are sent when they are.

The demo will illustrate the look-and-feel of VTView and VTSim by considering the well-known Alternating Bit Protocol modeled as a sender and receiver process communicating over an unreliable medium, also modeled as a process.

3 A Case Study: The i-protocol

The Concurrency Factory's local model checker was used to detect and diagnose a non-trivial livelock in the i-protocol, a bidirectional sliding-window protocol employed in the publicly available GNU UUCP file transfer utility. The model checker was dispatched on an instance of the protocol having a window size of 2, and explored about 1.079×10^6 states out of a total estimated global state space of 1.473×10^{12}.

Key to the Factory's success was the use of an abstraction to reduce the message sequence number space from 32 — the constant defined in the protocol's C-code — to $2W$, where W is the window size. This abstraction is shown to preserve the truthhood of all modal mu-calculus formulæ.

4 Conclusions

We have described the Concurrency Factory, a graphical environment that supports the following system development tasks: specification (VTView), simulation (VTSim), verification (model and bisimulation checking), and implementation (Facile graphical compiler). Much work remains to be done on the project, including better state-space management techniques and broader support for real-time systems.

5 References

[CS93] R. Cleaveland and B. U. Steffen. A linear-time model checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design*, 2, 1993.

[GMP89] A. Giacalone, P. Mishra, and S. Prasad. Facile: A symmertric integration of concurrent and functional programming. *International Journal of Parallel Programming*, 18(2), 1989.

[lic94] *Ninth Annual Symposium on Logic in Computer Science (LICS '94)*, Versailles, France, July 1994. Computer Society Press.

[Sok96] O. Sokolsky. *Efficient Graph-Based Algorithms for Model Checking in the Modal Mu-Calculus.* PhD thesis, Department of Computer Science, SUNY at Stony Brook, April 1996.

[SS94] O. Sokolsky and S. A. Smolka. Incremental model checking. In *Proceedings of the 6th International Conference on Computer-Aided Verification.* American Mathematical Society, 1994.

[SS95] O. Sokolsky and S. A. Smolka. Local model checking for real-time systems. In *Proceedings of the 7th International Conference on*

Computer-Aided Verification. American Mathematical Society, 1995.

[ZS92] S. Zhang and S. A. Smolka. Efficient parallelization of equivalence checking algorithms. In M. Diaz and R. Groz, editors, *Proceedings of FORTE '92 – Fifth International Conference on Formal Description Techniques*, pages 133–146, October 1992.