

Distributed Prototyping from Validated Specifications

Rance Cleaveland
SUNY at Stony Brook

Joint work with:
David Hansel (Reactive Systems Inc.)
Scott Smolka (Stony Brook)

Prototyping and Verification

The (rapid-)prototyping paradigm:

Generate *implementations* from *models*.

- E.g. CAPS, SYROCO, OpenCaesar, ...

The verification paradigm:

Analyze formal *models* for *properties*.

Why not use same models for both?

Prototyping from Formal Models?

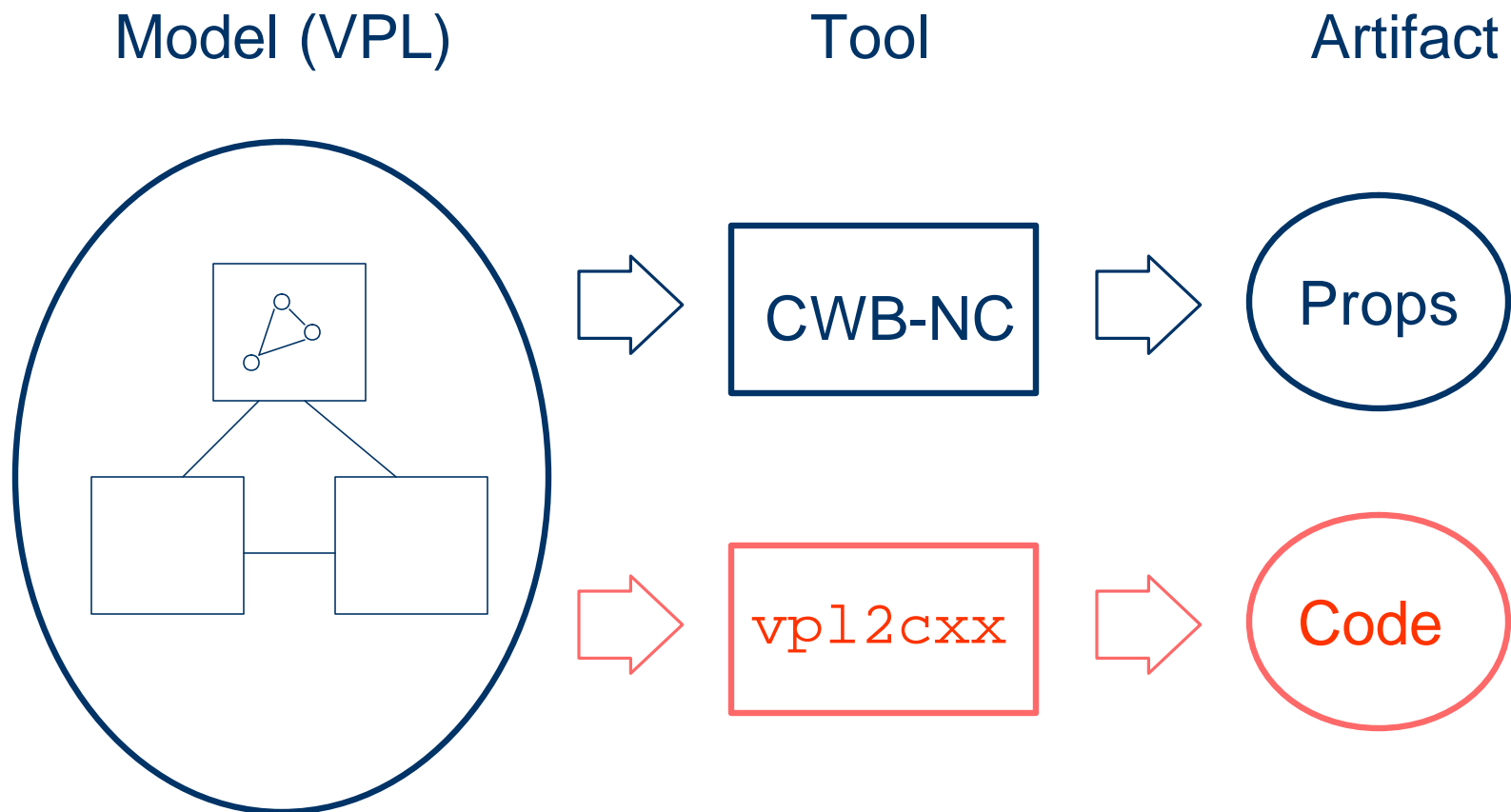
- Formal models can be mathematically validated
 - In practice, state explosion is a problem
- Prototypes can be tested
 - In practice, “coverage” is a problem
- Our perspective:
 - Analyze what you can on formal model ...
 - ... then generate prototype

This Paper

Distributed C++ prototypes from formal models

- Models given in process-algebra-inspired **Value Passing Language (VPL)**
- Models analyzable using **Concurrency Workbench of the New Century (CWB-NC)**
- Distributed C++ generated from VPL using communication libraries built on ACE

Overview



VPL

- Specification language for hierarchical systems of concurrent process
- Communication via message passing over typed channels
- Message passing: binary handshaking
- Formal semantics (SOS) defines translation from VPL models into state machines.

VPL: Example

```
value MAX:5
type t:MAX
channel c1:t
channel c2:t
```

```
process sender(chan: t)
begin
  var i:t
  i := 0;
  while (1=1)
    begin
      chan!i;
      if i=MAX-1 then
        i := 0
      else
        i := i + 1
      end
    end
  end;
end;
```

```
process buffer(inChannel:t,
              outChannel:t)
begin
  var buf:t
  while (1=1)
    inChannel?buf;
    outChannel!buf
  end
end;
```

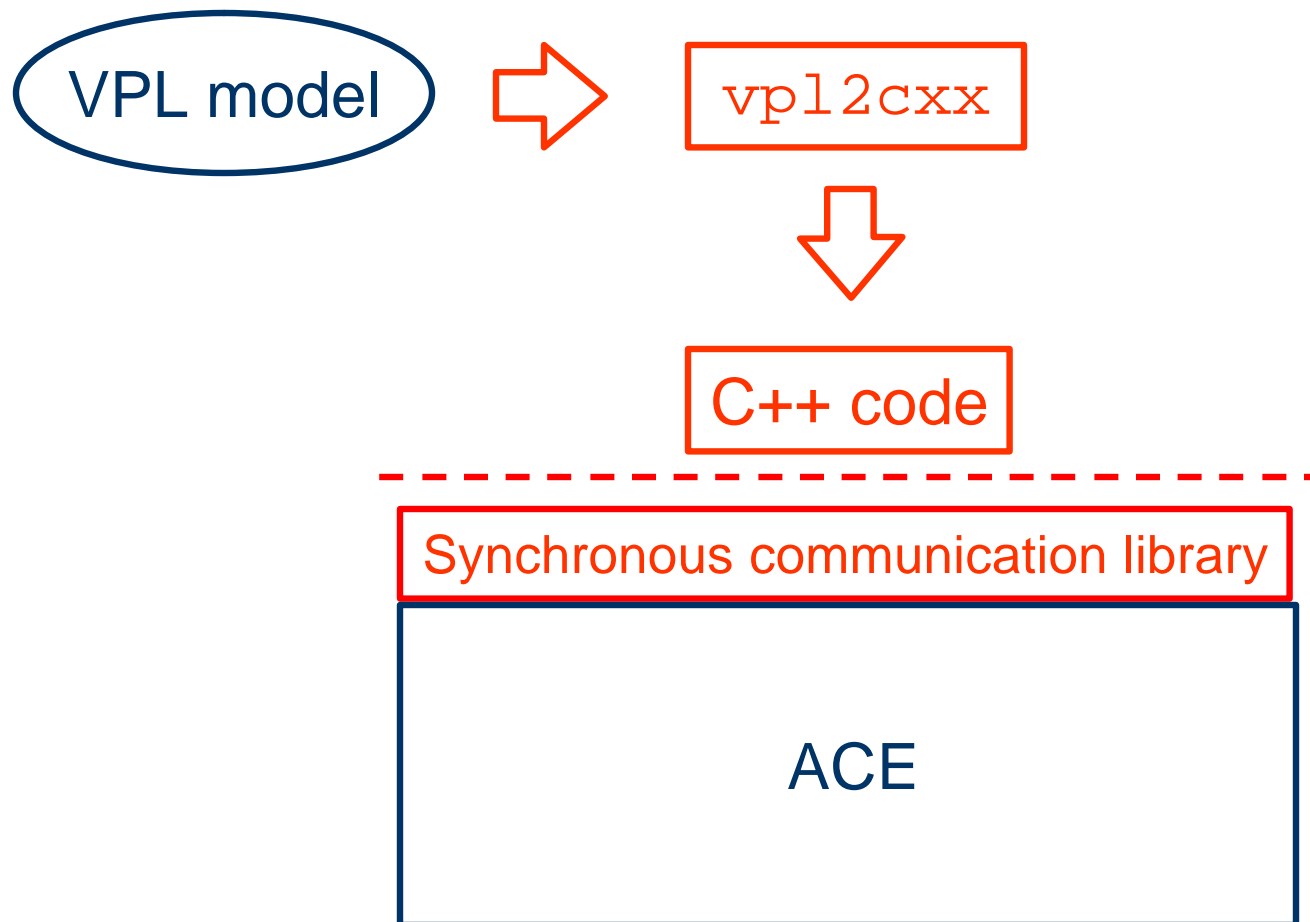
```
process receiver(chan: t)
begin
  var i:t
  while (1=1)
    chan?i;
  end
end;
```

```
sender(c1)
| buffer(c1,c2)
| receiver(c2)
```

vp12cxx

- Automatically generates C++ from VPL.
- Code produced is fully distributed, highly readable, and portable.
- Uses library for synchronous communication built on top of Adaptive Communication Environment (ACE).

vp12cxx Overview



vp12cxx in Detail

- Translates VPL models into C++ programs.
 - Code follows model structure to ensure readability.
 - Each process translated into different executable to ensure “distributedness”.
 - Synchronous communication implemented via calls to library that we wrote.
- Generates launch, configuration scripts.
- Communication library built on top of ACE to ensure portability.

ACE Network Programming Interface

- Developed by D.C. Schmidt at U.C. Irvine.
- Provides rich set of C++ wrappers for common communication tasks across a range of OS's.
- *Operating system adaptation layer* utilized by `vp12cxx` to achieve platform independence.

Synchronous Communication Library

- Methods for:
 - creating name & channel servers
 - client/server connections
 - channel I/O.
- Implements new randomized protocol for *I/O guard scheduling problem*.

The I/O Guard Scheduling Problem

- VPL processes can or-wait on send, receive.
- Fact: no fully distributed deterministic implementation can exist! [Francez]
- We developed *randomized* solution.
 - Processes “guess” which communication to attempt.
 - If guess is unsuccessful after an interval, processes “guess again”.
- Expected performance: $O(k^2)$ (k is number of “or choices”.)

The RETHER Case Study

- Software-based real-time ethernet protocol sold by RETHER Networks, Inc.
- Provides guaranteed bandwidth to multimedia applications via commodity ethernet hardware.
- We modeled RETHER in VPL and verified bandwidth guarantees and non-starvation.
- Then we applied `vp12cxx` to VPL spec.

Performance

- Preliminary study: `vp12cxx`-generated RETHER 10x slower than actual system.
 - Experiment involved four nodes, random traffic.
 - Difference due to `vp12cxx` code running in user, rather than kernel, mode.
- More experiments needed, but performance seems reasonable for prototyping purposes.

Conclusions and Future Work

- `vp12cxx` generates fully distributed prototypes from formal models of concurrent systems.
 - Generated code is readable, portable, efficient.
 - Implementation of communications library required solution to I/O guard scheduling problem.
- Future work:
 - More experimentation.
 - Extensions to VPL (real-time, other communication primitives, interoperability with other models)