Supporting Timed CSP Operators in CSP++

William B. Gardner & Yuriy Solovyov

Modeling & Design Automation Group
School of Computer Science
University of Guelph, Ontario, Canada
Outline

1. Overview of CSP++
2. Adding timed operators to CSP++
   - Verification and validation approaches
   - Translator, run-time framework, performance
3. Case study
4. Conclusion & future plans
5. Obtaining open source CSP++, contributing
1. Overview of Approach

CSP
formal “backbone”
models control structure
interprocess sync/comm
verification tools

CSP++
implements CSP computation model
invokes plug-in modules
handles interprocess sync/comm

Bridge between CSP and popular programming language

C++
plug-in modules
bulk of data processing
external I/O
restrictions (no IPC)
Notion of “selective formalism”

- Designer decides *how much* of system to model in CSP vs. C++
- Conceptual line between formal high-level spec and lower-level programming realization
  - move line “down” to enforce more rigorous formal modeling
  - move line “up” for reasons of efficiency or richness of language constructs
  - CSP not intended as full-featured prog. language
CSP++ Design Flow

CSP++ Design Flow:

1. **CSP Specs**
   - CSP Specifications

2. **Verification Tool**
   - CSP Verification Tool

3. **cspt translator**
   - Translation from CSP to CSP++

4. **CSP++ Control Layer**
   - User-coded Functions
   - Target System

CSP++ Design Flow:

- CSP Specifications are translated into CSP++ using the cspt translator.
- The translated CSP++ is used in the control layer, which includes user-coded functions and the target system.
Integration of User Code

CSP++ Control Layer

P1 CSP process
f1 CSP event

External Event Routines

P1 CSP process

RTOS Facilities

Hardware Components

Packages

f1 f2

P2

P3

f3 f4

user function
Restrictions on User Code

- Can link to individual events, or multiple cooperating events of leaf-level process
- Cannot rely on static storage (due to multiple process instances) except as could be provided by framework (*future work*)
- Cannot “go behind back” of CSP spec to contact other processes
  - preserves convention that interprocess communication/synchronization done via CSP
Related work

• NOCC compiler translates MCSP to execute on KRoC runtime [Barnes 2006]
• Component libraries with CSP semantics
  – JCSP/CCSP/C++CSP2; CTJ/CTC++; JACK
• [Raju et al 2003] translates CSPm to CTJ, JCSP, CCSP
  – CSP++ supports more operators ➔
Table 1. Translation support for FDR2's CSPm

<table>
<thead>
<tr>
<th>FDR2's CSPm Features</th>
<th>CSP-to-CTJ</th>
<th>CSP-to-JCSP</th>
<th>CSP++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments: --</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Comments: {- ... -}</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Integer data</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Declarations</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Process definitions</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Recursive processes</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Parameterized processes: P(2,1)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Prefix: =&gt;</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Chan?data, chan!data</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Chan?d1.d2..., chan!d1.d2...</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>If ... then ... else ...</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>External choice (alternative): []</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Interface (sharing) parallel: [][{...}]</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Interleaving parallel: P</td>
<td></td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>Sequential composition: P;Q</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Event renaming: [[e&lt;=f]]</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Event hiding: {e}</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Note (1): not needed for synthesis (treated as one-line comments)

Not supported

- Boolean guard: &
- Replicated operators: @
- Untimed timeout: [>
- Linked and alphabetized parallel
- Interrupt: \\
2. Adding timed operators to CSP++

- Original motivation: modeling of financial transactions
- Modeling of “soft” real-time systems
  - Not safety-critical, where timing constraints must be guaranteed
- Had planned to support untimed interrupt \(\land\) and timeout \([>)\) — may as well add timed counterparts
Verification approaches

• For untimed portions of spec, or where timing does not affect sync with other processes, remove time constants and use FDR2 as usual

• Where timing is important, can use HORAE tool [Dong et al 2006, Nat. Univ. Singapore], minimal syntax difference from CSP++

• *New option:* convert timed operators to tock equivalent, and use FDR 2.94 feature that allows processes to sync on tock without resolving choice
Post-run trace validation

- Run program with –t option to output trace
- Python script available to format and send trace to FDR2 to check trace refinement of CSPm spec
5 new operators

• Timed prefix: \( a \xrightarrow{-5} b \xrightarrow{-2} \text{SKIP} \)
  – At least \( t \) time units will elapse before next event

• Timed timeout: \( a \rightarrow P \ [10] Q \)
  – Give a \( t \) time units to start, else continue as \( Q \)
    • First event \( a \) should be subject to synchronization

• Untimed timeout: \( P \ [e] \rightarrow Q \)
  – Event \( e \) will preempt \( P \) from starting
5 new operators (cont.)

• Untimed interrupt: $P \setminus e \rightarrow Q$
  – Event e will grab control from unfinished P

• Timed interrupt: $P \setminus 8 \setminus Q$
  – P has $t$ time units to finish, else Q grabs control
    • Not like operating system interrupt!

Notes:

– Interrupt applies to all subprocesses of P
– Set time unit by pragma or run-time option
  • msec, second, minute, hour
Timed prefix

• Implementation
  – Translator generates a call to framework function to make thread sleep for $t$ time units

• Special considerations
  – In case process is in scope of interrupt operator, timed wait must be interruptible
    • GNU Pth allows this
Timeouts

• Both timeouts treated as a kind of deterministic choice: $a \rightarrow P \ "[\]" \ e \rightarrow Q$
  – If event a succeeds (does not block), P wins and the timeout to Q does not occur

• Timed version: $a \rightarrow P \ \{10\rightarrow Q$
  – Limit blocking wait for event a to $t$ time units (interruptible like timed prefix blocking)
“Untimed” timeout

• Untimed version: $a \rightarrow P \ [> \ Q$
  – Try a first; if not succeed, resolve choice to Q

• A valid and useful “polling” interpretation
  – Different from regular choice $a \rightarrow P \ [ \ b \rightarrow Q$
    • CSP++ tries alternatives from left to right anyway
    • Normally, if a and b don’t succeed, keeps waiting for both, but in [> case, if a does not succeed, it loses its chance and “times out” to b->Q
Interrupts

• **Main challenge**: extricating thread of control from interrupted process so that...
  – it does not contribute any more events to the system trace following the interrupting event
  – all internal data structures are cleaned up

• **Key method**: interrupting event triggers interrupted process to throw C++ exception
  – CSP++ avoided exceptions for fear of overhead
Implementing interrupts

\[ S = P \land e \rightarrow Q \]

- Translator generates code to push EnvInt object on S’s environment stack
  - Acts as control centre for that interrupt
  - Nested interrupt operators work as well!

- Event e is tried first:
  - If succeeds, P never starts, S continues as Q
  - Else, spawns thread for P, and S waits for e
Interrupts (cont.)

- P’s events executing under scope of EnvInt environment object check its flag to see if interrupt occurred
  - If so, P throws exception, caught at “top” of thread, which cleans up and exits
- If P finishes without event e occurring, the EnvInt object is popped off and S terminates (or carries on) normally
Performance impact

• **Was fear of C++ exceptions justified?**
  – “NO” (at least for g++)

• **Additional execution time and memory costs were only around 1%**
  – Negligible cost if no interrupts coded in spec
  – Highest cost to execute processes within scope of interrupt operator (checking flags, etc.)
3. “VAC” case study

- Robot vacuum cleaner demonstrates all new operators
VAC interrupts

ROBOT(1) =

RUNNING /20\ low_battery -> SHUTOFF

If robot does not complete RUNNING process within 20 time units, it will cause a low_battery event and go into SHUTOFF.

RUNNING =

WHICHOPMODE \ pickup -> EMERGENCY_STOP

While running normally within WHICHOPMODE process, if a sensor detects a pickup event (by the human), it will immediately go into EMERGENCY_STOP.
The process executes a series of checks:

- If it detects the **adone** event, it pauses one time unit and terminates.

- If not, it checks for **dust** and **clean**s it, then loops back.

- If no dust, it **idles** for one time unit, then loops back.
WHICHOPMODE = 

  (manual -> REMOTE_CONTROL) [>
  ((turn_off -> ROBOT(0)) [7>
    AUTOMATIC_MODE) 

The process checks for the **manual** mode event, and if succeeds, enters REMOTE_CONTROL.
Otherwise, it waits up to 7 time units for a **turn_off** event, which will put it into ROBOT(0). But if the timeout expires, it will default into AUTOMATIC_MODE.
4. Conclusion & Future Plans

- CSP++ makes synthesizable subset of \textit{timed} CSPm specifications executable & extensible
  - Useful for pedagogy $\rightarrow$ CSPm simulator
  - Tool for carrying out selective formalism with user-coded C++ functions
  - Possibility of making (some) formalism more palatable & practical to the resistant
Future plans

• Work underway…
  – Making selective formalism more practical by providing mechanism for UCFs to access “process-specific storage” with managed scope
  – Garner & Roggenbach (Swansea), adding data types (sequence, set) and inline functions

• Future work includes…
  – Replicated operators (@), interruptible UCFs
5. Open source project!

- CSP++ home page
  - www.uoguelph.ca/~gardnerw/csp++
  - Licenses: translator GPL, run-time framework LGPL (can use to build proprietary system)

- Contributors welcome!