The Discrete Time ToolBus

Arie van Deursen
Eindhoven University of Technology

Programme:

- The ToolBus (45 min)
- Break (15 min)
- Discrete Time Extensions (30 min)
- Demo (HG7.25 / HG7.01) (15 min)
Coupling Software Components

- How to build software that is large, heterogeneous, and distributed?

- How to connect a number of independent, interactive tools?

- How to integrate tools into a well-defined, cooperating system?

- How to decompose a single monolithic system into a number of separated exchangeable components?
Integration Problems

- Start several components (Unix processes), exchange data, stop components, connect new ones, etc.

- **Data Integration**
  
  Software components must agree on exchange format for shared data.

- **Control Integration**
  
  Who is doing what in which order?

  When and how is data communicated?
The ToolBus Architecture

- **ToolBus**: Hardware metaphor: Tools are connected via a software *bus*.

- Tools can *only* communicate via bus.

- System builders can write a *ToolBus script*: Manages communication between tools, and their parallel, sequential or iterative composition.

- Script: written using *process algebra* formalism based on (discrete time) ACP.

- ToolBus: Script interpreter.
Atomic Actions

- Print a string:
  `printf( <String> )`

- Starting external tools:
  `execute( <String>, <Pid?> )`

- Request tool to do something:
  `snd-do( <Pid>, <Request> )`

- Receive event from a tool:
  `rec-event( <Pid>, <Event> )`

- Receive a request to get attached:
  `rec-connect( <Pid?> )`

- Atomic actions take *no* time!!
Component Adapters

• ToolBus implemented using Unix sockets (shared memory, pipes, TCP/IP).

• Components written in C, Tcl/Tk, Perl, ...

• Required:
  – Mapping from ToolBus data terms to component-specific data structure
  – Translation of component activities to ToolBus events.

• Needed: language or tool specific adapters. (Possibility to generate partly from ToolBus script).
A Simple Calculator

- Two tools:
  - A calculator which can compute expressions like $3 \times (4 + 6)$.
  - A user interface process which just displays buttons for 0...9, +, and *.

- Connected by ToolBus:

  Start calculator and user interface;
  Then: wait for input, send input from ui to calc, and ask to compute.

  Two outcomes:
  OK: then display the value;
  otherwise: ask ui to re-edit original expression.
A ToolBus Calculator

tool calc is {command "/home/arie/bin/calcul"}
tool ui is {command "wish-adapter -script ui-calculator.tcl"}

process CALC-CONNECTOR is
let CalcId: calc,
   UId: ui,
   E: str,
   V: int
in
   ( execute(ui, UId?) || execute(calc, CalcId?) ) .
   ( rec-event(UId, request-computation, E?) ) .
   snd-eval(CalcId, compute(E)) .
   ( rec-value( CalcId, result(V?) ) .
     snd-do( UId, display-value(V))
     +
     rec-value( CalcId, error-value ) .
     snd-do( UId, retry(E))
   )
) * delta
endlet

toolbus(CALC-CONNECTOR)
State Operator

- The let construct introduces a local state.
- The state can be adapted by assignments.

- Axiomatization:

\[
\begin{align*}
\lambda_S(\delta) &= \delta \\
\lambda_S(a) &= \text{action}(a, S) \\
\lambda_S(a \cdot x) &= \text{action}(a, S) \cdot \lambda_{\text{effect}(a, S)}(x) \\
\lambda_S(x + y) &= \lambda_S(x) + \lambda_S(y)
\end{align*}
\]

- \( S \): the state that is changed;
  \text{action}(a, S): renamed action for \( a \);
  \text{effect}(a, S): state change caused by \( a \).
Data Terms

• Tools and ToolBus exchange data in the form of terms.

• Syntactic forms:

\[
\begin{align*}
\text{Var} & : = [A-Z][A-Za-z0-9]* \\
\text{Id} & : = [a-z][A-Za-z0-9]* \\
\text{Str} & : = '^[']*'^' \\
\text{Int} & : = '-'? [0-9]+ \\
\text{Term} & : = \text{Id} | \text{Var} | \text{Var }'?'|| \text{Str} | \text{Int} \\
& \quad : = | \text{Id }'(' \{ \text{Term },' \}+ ')'
\end{align*}
\]

• Examples:
  
  \[
  \begin{align*}
  & \text{plus}(3, \text{times}(4,7)) \\
  & \text{request-computation} \\
  & \text{retry}("3 + (4 * 7)"") \\
  & \text{retry}(E)
  \end{align*}
  \]
Matching and Communication

- Data terms are exchanged, e.g., by rec-value, snd-do, rec-event atoms.

- Open terms: contain variables with question mark.

- Value passing: match terms, and assign variables if successful.

- Examples:
  
  plus(3,times(4,7))   plus(E?)   E := times(4,7)
  req-computation     req-reslt   δ
  plus(3,F)           plus(E?)   E := F
Iteration

● Iteration by binary Kleene star:

\[ x \cdot y = x \cdot (x \cdot y) + y \]

● Infinite loop:

\[ x \cdot \text{delta} \]

● while \( E \) do \( x \) od; \( y \)

\[ \text{if } E \text{ then } x \text{ fi } \ast \text{ if } \neg E \text{ then } y \text{ fi} \]

● ToolBus has no recursive process definitions.
More Complicated Scripts

• If tool communication gets more complicated, its description in ACP can be split into several processes.

• Process definitions can be parameterized by data.

• (Synchronous) communication is possible using snd-msg(...) and rec-msg(...)

• Asynchronous communication is possible by broadcasting notes.

• Processes can subscribe to notes of certain types.

• No recursion (use iteration)
Applications

- Redesign of specification development environment (link parser, editors, type checker, reduction machine, pretty printer, theorem prover, ...)

- Traffic light protocol (Nederland Haarlem)

- ACP interpreter / simulator, using viewer / printf facilities.

- Distributed “zeeslag”, using Tcl/Tk
Discrete Time ToolBus

- ToolBus: complicated piece of software.

- A formal description was given using algebraic specification (the ASF+SDF formalism and system)

- ToolBus supports a wide variety of features; necessary to describe in a modular way.

- Latest addition: simple form of discrete time.

- Possibility to use delay and time outs.
Discrete Time

- Timed actions: postfixed by timer information.

- Timer postfixes:
  - $\text{delay}(\text{sec}(10))$ wait 10 seconds
  - $\text{abs-delay}$
  - $\text{timeout}(\text{sec}(10))$ act within 10 seconds
  - $\text{abs-timeout}$

- Either all absolute or all relative.
Example: Distributed Auction

process OneSale(Item: str, Initial: int) is
  let Final: Bool, Sold: bool, Highest: int
  in
    ( Final:=false || Sold:=false || Highest:=Initial ).
    while not(Sold) do
      rec-msg(bid(Bidder?, Amount?)) .
      if greater(Amount, HighestBid)
        then HighestBid := Amount || Final := false fi
      +
      if not(Final) then
        snd-note(any-higher-bid) delay(sec(10)) .
        Final := true
      fi
      +
      if Final then
        snd-note(sold(HighestBid)) delay(sec(10))
        Sold := true
      fi
    ) * if Sold then snd-ack-event( ... )
endlet
Specification

- Formal description of the ToolBus: time encoded in state. (Sec. 6.10)

- State has a field *current-time*, which can be updated and inspected.

- Every action can add one to the current-time

- In transition graph, every node can do a time step and get back to itself.

- Timer functions: expanded to conditionals on current-time.
Implementation

- Time updates only at clock ticks.

- At each moment, compute which actions can be performed.

- Check for occurrences of timer functions.

- Look for nearest time, let seconds elapse, and then make the choice.
Axiomatization

• Given in Appendix E of ToolBus report.

• $(a)(n + 1)$: *must* perform $a$ in slice $n + 1$

• $a^\vee(n + 1) = (a)(n + 1) + \delta$.

• $k \gg x$: initialize $x$ at time slice $k$.

• $a^\vee(k + 1) \cdot x = a^\vee(k + 1) \cdot (k \gg x)$

• $a^\vee(k) \ (l \gg x) = a^\vee(k) \cdot (l \gg x)$
Applications and Future Developments

• Distributed auction: Use timers for eenmaal, andermaal, ...

• ”Compact dynamisch busstation Apeldoorn.”, similar to busstation of Eindhoven.
  
  Bus platforms should not be unused, timed arrival of busses.

  Complicated mixture of ToolBus, Perl, and Tcl/Tk.

• In near future: develop more adapters (COBOL, Standard ML)

• Applications in software renovation.