Core AUTOSAR Software Components as a simple calculus

Johan Nordlander, Sept 2013

Purpose

- To clarify the semantics of AUTOSAR SWCs without some of the detail inherent in a simulator implementation
- To serve as a starting-point for a simulator implementation
- To obtain a light-weight framework for further experiments in AUTOSAR formalization

Approach

- A simple process calculus with <u>broadcast</u> communication in the style of CBS (good match with AUTOSAR's frequent use of publish-subscribe patterns)
- <u>Static</u> scoping and process structure (following AUTOSAR)
- Currently limited to a <u>flat</u> process hierarchy (any AUTOSAR component hierarchy can be transformed into a flat one)

Names

- Identify component instances, runnables, ports, data elements, server operations, ...
- AUTOSAR guarantees:
  - component instance names *i* are globally unique
  - runnables *r* & ports *p* are unique within an instance
  - elements *e* & operations *o* are unique within a port
  - etc
- Leads to <u>hierarchical</u> names:

  i.s (inter-runnable vars) *i.x* (exclusive areas) *i.r* (runnables)
  i.p (ports) *i.p.e* (data elements) *i.p.o* (server ops)
- Union of all names ranged over by *a*, *b* and *c*

Process terms

- Grammar:

$$
\begin{array}{lll}
P, Q & ::= & a \triangleright A \quad | \quad P \parallel Q \quad | \quad 0 \\
A & ::= & \text{named atomic processes (next)}
\end{array}
$$

- Labelled reduction:

$$P \xrightarrow{e} P'$$

- Labels: *(hear)* *(say)* *(age)*

$$
\begin{array}{lll}
e, f & ::= & a?L \quad | \quad a!L \quad | \quad \partial_t \\
L & ::= & \text{label payload (to follow)}
\end{array}
$$

Atomic processes

$i.r \rhd \mathrm{Run}\{time, act, n\}$	Common state for runnable *r* of component *i*, showing *n* current instances, *time* seconds since last activation, and activation state *act*
$i.r \rhd \mathrm{RunInst}\{c, ex, code\}$	An instance of runnable *r* within component *i*, currently executing *code* and owning exclusive area set *ex*, possibly on behalf of client *c*
$i.x \rhd \mathrm{Excl}\{bool\}$	Exclusive area *x* of component instance *i*, with current busy state

Atomic processes

$i.s \rhd \mathrm{Irv}\{value\}$	Inter-runnable variable s of component i, with a current $value$
$i.p.e \rhd \mathrm{QElem}\{n, values\}$	Queued data element of size n holding a sequence of $values$
$i.p.e \rhd \mathrm{DElem}\{upd, value\}$	Data element holding a single $value$ with an update flag upd
$i.p.o \rhd \mathrm{Op}\{values\}$	Client-side operation buffer, holding a sequence of return $values$
$i.r \rhd \mathrm{Timer}\{time\}$	Timer for runnable $i.r$ with $time$ seconds left

Parallelism & broadcast

Parallel reduction:

$$\frac{P \xrightarrow{e} P' \qquad\qquad Q \xrightarrow{f} Q'}{P \mathbin{||} Q \xrightarrow{e \bullet f} P' \mathbin{||} Q'}$$

where • is a partial label combinator:

$$a?L_1 \bullet a!L_2 = a!(L_1 \sqcup L_2)$$

$$a!L_1 \bullet a?L_2 = a!(L_1 \sqcup L_2)$$

$$a?L_1 \bullet a?L_2 = a?(L_1 \sqcup L_2)$$

$$\partial_t \bullet \partial_t = \partial_t$$

The code of a runnable

code	::	*Code*	
code	::=	Send(*p*,e,*v*,*cont*)	Send to a QElem
	|	Receive(*p*,e,*cont*)	Read from a QElem
	|	Write(*p*,e,*v*,*cont*)	Write to a DElem
	|	Read(*p*,e,*cont*)	Read from a DElem
	|	IsUpdated(*p*,e,*cont*)	Check flag of a DElem
	|	Invalidate(*p*,e,*cont*)	Empty a DElem
	|	Call(*p*,*o*,*v*,*cont*)	Invoke a server runnable
	|	Result(*p*,*o*,*cont*)	Fetch result of previous Call
	|	IrvWrite(*s*,*v*,*code*)	Write to an Irv
	|	IrvRead(*s*,*v*,*cont*)	Read from an Irv
	|	Enter(*x*,*code*)	Acquire an exclusive token
	|	Exit(*x*,*code*)	Return an exclusive token
	|	Terminate(*v*)	Terminate

Values & results

- Values *v* range over standard C values (including arrays, structs and unions)
- Observable operations return values of type *Std_ReturnType*, which is the disjoint union of proper values and a set of error tokens
- A code continuation (ranged over by *cont*) is a function from *Std_ReturnType* to *Code*

Runnable attributes

- Dynamic semantics is defined relative to the static info in a given AUTOSAR system model (see ARText.hs for abstract syntax)
- Most relevant static info is a set of attributes for each runnable, with names like events, canBeInvokedConcurrently, minimumStartInterval, etc
- Static attribute attr of runnable *i.r* is here referred to as attr(*i.r*)

Port interconnections

- Also part of the static AUTOSAR model info
- Captured as a relation $\Rightarrow$ between names:
  - $i.p \Rightarrow i'.p'$ iff there is the model connects port p of component i to port p' of component i'
  - Lifted to $i.p.e \Rightarrow i'.p'.e$ for all elements e of connected sender-receiver ports $i.p$ and $i'.p'$
  - Ditto for all operations of client-server ports
  - Ditto for all port delegations of the (root) component composition

Reduction axioms

- Constitute the core of the dynamic semantics
- Appear in matching groups that define the ways of <u>saying</u> as well as <u>hearing</u> a particular broadcast payload
- A separate set of axioms define how atomic terms react to the passage of time (label ∂_t)
- The balance between time-steps and proper work is not fixed here ("speed-agnosticism")

Initial semantic state

A parallel composition of:

- For each component prototype of the (top-level) composition:
  - One Excl term for each exclusive area
  - One Run term for each runnable
  - One Irv term for each inter-runnable variable
  - For each <u>required</u> sender-receiver port:
    - one QElem term for each QueuedComSpec element
    - one DElem term for each UnQueuedComSpec element
  - For each <u>required</u> client-server port:
    - One Op term for each operation
  - One Timer term for each Timing event of each runnable

Miscellaneous

- Sequences (always flat) are written using : for both left and right concatenation
- An activation state *act* for a runnable with an OpInvokedEvent is of the form Serving{*clients*,*args*}, where *clients* and *args* are sequences
- Otherwise, *act* toggles between Idle and Pending
- Initial *act* values are either Serving{[],[]} or Idle

Exclusive areas

$$i.r \rhd \text{RunInst}\{c, ex, \text{Enter}(x, code)\} \xrightarrow{i.x!\text{enter}()} i.r \rhd \text{RunInst}\{c, x{:}ex, code\}$$

$$a \rhd \text{Excl}\{True\} \xrightarrow{a?\text{enter}()} a \rhd \text{Excl}\{False\}$$

$$i.r \rhd \text{RunInst}\{c, x{:}ex, \text{Exit}(x, code)\} \xrightarrow{i.x!\text{exit}()} i.r \rhd \text{RunInst}\{c, ex, code\}$$

$$a \rhd \text{Excl}\{False\} \xrightarrow{a?\text{exit}()} a \rhd \text{Excl}\{True\}$$

Inter-runnable variables

$$i.r \rhd \text{RunInst}\{c, ex, \text{IrvRead}(s, cont)\} \xrightarrow{i.s!\text{irvr}(v)} i.r \rhd \text{RunInst}\{c, ex, cont(v)\}$$

$$a \rhd \text{Irv}\{v\} \xrightarrow{a?\text{irvr}(v)} a \rhd \text{Irv}\{v\}$$

$$i.r \rhd \text{RunInst}\{c, ex, \text{IrvWrite}(s, code)\} \xrightarrow{i.s!\text{irvw}(v)} i.r \rhd \text{RunInst}\{c, ex, code\}$$

$$a \rhd \text{Irv}\{\_\} \xrightarrow{a?\text{irvw}(v)} a \rhd \text{Irv}\{v\}$$

Sending/receiving

$$i.r \rhd \mathrm{RunInst}\{c, ex, \mathrm{Receive}(p,e,cont)\} \xrightarrow{i.p.\mathrm{e}!\mathrm{rcv}(v)} i.r \rhd \mathrm{RunInst}\{c, ex, cont(v)\}$$

$$a \rhd \mathrm{QElem}\{n, v{:}vs\} \xrightarrow{a?\mathrm{rcv}(v)} a \rhd \mathrm{QElem}\{n, vs\}$$

$$a \rhd \mathrm{QElem}\{n, []\} \xrightarrow{a?\mathrm{rcv}(\mathrm{NO\_DATA})} a \rhd \mathrm{QElem}\{n, []\}$$

$$i.r \rhd \mathrm{RunInst}\{c, ex, \mathrm{Send}(p,e,v,cont)\} \xrightarrow{i.p.\mathrm{e}!\mathrm{snd}(v,res)} i.r \rhd \mathrm{RunInst}\{c, ex, cont(res)\}$$

$$a \rhd \mathrm{QElem}\{n, vs\} \xrightarrow{b?\mathrm{snd}(v,\mathrm{OK})} a \rhd \mathrm{QElem}\{n, vs{:}v\} \quad \text{if } b \Rightarrow a \;\&\; |vs| < n$$

$$a \rhd \mathrm{QElem}\{n, vs\} \xrightarrow{b?\mathrm{snd}(v,\mathrm{LIMIT})} a \rhd \mathrm{QElem}\{n, vs\} \quad \text{if } b \Rightarrow a \;\&\; |vs| = n$$

$$i.r \rhd \mathrm{Run}\{t, \_, n\} \xrightarrow{b?\mathrm{snd}(v,v')} i.r \rhd \mathrm{Run}\{t, \mathrm{Pending}, n\}$$

$$\text{if } b \Rightarrow i.p.\mathrm{e} \text{ and } \mathrm{DataReceived}(p.\mathrm{e}) \in \mathrm{events}(i.r)$$

Reading/writing

$$i.r \rhd \mathrm{RunInst}\{c, ex, \mathrm{Read}(p,e,cont)\} \xrightarrow{i.p.\mathrm{e!rd}(v)} i.r \rhd \mathrm{RunInst}\{c, ex, cont(v)\}$$

$$a \rhd \mathrm{DElem}\{u, v\} \xrightarrow{a?\mathrm{rd}(v)} a \rhd \mathrm{DElem}\{\mathrm{False}, v\}$$

$$i.r \rhd \mathrm{RunInst}\{c, ex, \mathrm{Write}(p,e,v,cont)\} \xrightarrow{i.p.\mathrm{e!wr}(v)} i.r \rhd \mathrm{RunInst}\{c, ex, cont(\mathrm{OK})\}$$

$$a \rhd \mathrm{DElem}\{u, \_\} \xrightarrow{b?\mathrm{wr}(v)} a \rhd \mathrm{DElem}\{\mathrm{True}, v\} \qquad \text{if } b \Rightarrow a$$

$$i.r \rhd \mathrm{Run}\{t, \_, n\} \xrightarrow{b?\mathrm{wr}(v)} i.r \rhd \mathrm{Run}\{t, \mathrm{Pending}, n\}$$

$$\text{if } b \Rightarrow i.p.\mathrm{e} \text{ and } \mathrm{DataReceived}(p.\mathrm{e}) \in \mathrm{events}(i.r)$$

Reading/writing

$$i.r \rhd \text{RunInst}\{c, ex, \text{IsUpdated}(p,\text{e},cont)\} \xrightarrow{i.p.\text{e}!\text{up}(u)} i.r \rhd \text{RunInst}\{c, ex, cont(u)\}$$

$$a \rhd \text{DElem}\{u, v\} \xrightarrow{a?\text{up}(u)} a \rhd \text{DElem}\{u, v\}$$

$$i.r \rhd \text{RunInst}\{c, ex, \text{Invalidate}(p,\text{e},cont)\} \xrightarrow{i.p.\text{e}!\text{inv}()} i.r \rhd \text{RunInst}\{c, ex, cont(\text{OK})\}$$

$$a \rhd \text{DElem}\{u, \_\} \xrightarrow{b?\text{inv}()} a \rhd \text{DElem}\{\text{True}, \text{INVALID}\}$$

if $b \Rightarrow a$

Calling a server

$i.r \rhd \text{RunInst}\{c, ex, \text{Call}(p,o,v,cont)\} \xrightarrow{i.p.o!\text{call}(v,res)} i.r \rhd \text{RunInst}\{c, ex, cont(res)\}$

if $\text{ASync}(p.o) \in \text{serverCallPoints}(i.r)$ or $res \neq \text{OK}$

$i.r \rhd \text{RunInst}\{c, ex, \text{Call}(p,o,v,cont)\} \xrightarrow{i.p.o!\text{call}(v,\text{OK})} i.r \rhd \text{RunInst}\{c, ex, \text{Result}(p,o,cont)\}$

if $\text{Sync}(p.o) \in \text{serverCallPoints}(i.r)$

$i.r \rhd \text{Run}\{t, \text{Serving}\{clients,vs\}, n\} \xrightarrow{c?\text{call}(v,\text{OK})} i.r \rhd \text{Run}\{t, \text{Serving}\{clients{:}c,vs{:}v\}, n\}$

if $c \Rightarrow i.p.o$ & $\text{OpInvoked}(p.o) \in \text{events}(i.r)$ & $c \notin clients$

$i.r \rhd \text{Run}\{t, \text{Serving}\{clients,vs\}, n\} \xrightarrow{c?\text{call}(v,\text{LIMIT})} i.r \rhd \text{Run}\{t, \text{Serving}\{clients,vs\}, n\}$

if $c \in clients$

Obtaining a server result

$$i.r \rhd \text{RunInst}\{c, ex, \text{Result}(p,o,cont)\} \xrightarrow{i.p.o!\text{res}(v)} i.r \rhd \text{RunInst}\{c, ex, cont(v)\}$$

$$a \rhd \text{Op}\{v{:}vs\} \xrightarrow{a?\text{res}(v)} a \rhd \text{Op}\{vs\}$$

$$a \rhd \text{Op}\{[]\} \xrightarrow{a?\text{res}(\text{NO\_DATA})} a \rhd \text{Op}\{[]\}$$

$$a \rhd \text{RunInst}\{i.p.o, [], \text{Terminate}(v)\} \xrightarrow{i.p.o!\text{ret}(v)} a \rhd \text{RunInst}\{\,.\,, [], \text{Terminate}(\text{VOID})\}$$

$$a \rhd \text{Op}\{vs\} \xrightarrow{a?\text{ret}(v)} a \rhd \text{Op}\{vs{:}v\}$$

Spawning and terminating

$a \rhd \text{RunInst}\{\,.\,,[\,],\text{Terminate}(\text{VOID})\} \xrightarrow{a!\text{term}()} 0$ (0 is silently consumed by ||)

$a \rhd \text{Run}\{t, act, n\} \xrightarrow{a?\text{term}()} a \rhd \text{Run}\{t, \text{act}, n\text{-}1\}$

$a \rhd \text{Run}\{0, \text{Pending}, n\} \xrightarrow{a!\text{new}()} a \rhd \text{Run}\{t, \text{Idle}, n{+}1\} \parallel a \rhd \text{RunInst}\{\,.\,,[\,], cont(\text{VOID})\}$

if $n = 0$ or canBeInvokedConcurrently(a), where
t = minimumStartInterval(a) and $cont$ = implementation(a)

$a \rhd \text{Run}\{0, \text{Serving}\{c{:}cs, v{:}vs\}, n\} \xrightarrow{a!\text{new}()} a \rhd \text{Run}\{t, \text{Serving}\{cs, vs\}, n{+}1\} \parallel a \rhd \text{RunInst}\{c, [\,], cont(v)\}$

if $n = 0$ or canBeInvokedConcurrently(a), where
t = minimumStartInterval(a) and $cont$ = implementation(a)

Passing time

$a \rhd \mathrm{Timer}\{0\} \xrightarrow{a!\mathrm{tick}()} a \rhd \mathrm{Timer}\{t\}$ if $\mathrm{Timing}(t) \in \mathrm{events}(a)$

$a \rhd \mathrm{Run}\{t, \_, n\} \xrightarrow{a?\mathrm{tick}()} a \rhd \mathrm{Run}\{t, \mathrm{Pending}, n\}$

$a \rhd \mathrm{Run}\{t, act, n\} \xrightarrow{\partial_v} a \rhd \mathrm{Run}\{t\text{-}v, act, n\}$ if $t \geq v$

$a \rhd \mathrm{Timer}\{t\} \xrightarrow{\partial_v} a \rhd \mathrm{Timer}\{t\text{-}v\}$ if $t \geq v$

$a \rhd A \xrightarrow{\partial_v} a \rhd A$ if $A \neq \mathit{Run}\{..\}$ & $A \neq \mathit{Timer}\{..\}$

Ignoring a broadcast

- For an atomic term, ignoring a broadcast means to hear but not to react – formally

$$a \rhd A \xrightarrow{b?L} a \rhd A$$

- However, it is important that terms do not discard broadcasts arbitrarily. Therefore, the rule above only applies if $a \neq b$ and $a \not\Rightarrow b$.

- *(Strictly speaking, since RunInst and Timer terms just reuse the names of their respective runnables, the above restriction should not apply to them – they can always ignore what they hear. Must formalize this in a better way...)*

Next...

- These definitions just mark the beginning, much remains to be done – both in terms of sanity-checking and additional features
- The goal is a simulator implementation rather than a theoretical study, though, so a Haskell encoding is what should follow next
- I'va already encountered several ambiguities in the AUTOSAR documents, which can be described and discussed using this formalism – I will assemble a list of issues shortly

References

- AUTOSAR Software Component spec
  http://www.autosar.org/download/R4.1/AUTOSAR_TPS_SoftwareComponentTemplate.pdf
- AUTOSAR Run-Time Environment spec
  http://www.autosar.org/download/R4.1/AUTOSAR_SWS_RTE.pdf
- ARText – textual syntax for AUTOSAR SWCs
  http://www.arccore.com/devon/help/index.jsp?topic=%2Forg.artop.artext.help%2Fdoc-gen%2FSoftware-Component-Language-.html&cp=6_2
- Haskell encoding of the ARText abstract syntax
  https://github.com/josefs/autosar/ARText.hs