GENERATING COMPOSITE BEHAVIOR OF EMBEDDED SOFTWARE COMPONENTS BASED ON UML BEHAVIOURAL MODEL AND PROCESS ALGEBRA

2010.11.16
Jinhyun Kim
Korea University
Contents

• Introduction
• Background
• TRoS and ACSR
• TRoS || ACSR for Embedded Software Components
• Specifying Real-time Embedded Applications in TRoS || ACSR
• Conclusions
Contents

• Introduction
  • Background
  • TRoIS and ACSR
  • TRoS || ACSR for Embedded Software Components
  • Specifying Real-time Embedded Applications in TRoS || ACSR
  • Conclusions
PLC for NPP
STATEMATE Statecharts

- It is widely used to model not only HW but also embedded SW,
- There are plenty of analyzing tools, such as model checking, simulation, and test
Model of pCOS
Semaphore Model in SC
Semaphore in ACSR

```
SemMng_B_SEM_1
 = {}:SemMng_B_SEM_1
 + (OSSemCreate_B_SEM_1,1).B_SEM_1_Pend ;

B_SEM_1_Pend
 = {}:B_SEM_1_Pend
 + (OSSemPend_SEM_1,1).B_SEM_1_Post
 + (OSSemDelete_B_SEM_1,1).SemMng_B_SEM_1;

B_SEM_1_Post
 = {}:B_SEM_1_Post
 + (OSSemPost_SEM_1,1).('OSSched,1).B_SEM_1_Pend
```
Contents

- Introduction
- Background
- TRoS and ACSR
  - TRoS || ACSR for Embedded Software Components
  - Specifying Real-time Embedded Applications in TRoS || ACSR
- Conclusions
TRoS

- TRoS = Statecharts + Timed Actions
- The timed action specifies timed behaviors constrained by the prioritized use of resources.
- It can be transformed directly from ACSR in easy way by rules we have defined.
Timed Action

- \( A ::= \{S\}^n \mid \langle S \rangle^n \)
- \( S ::= \epsilon \mid (r,ve),S \)
Example
Algorithm 1 TRoSExec

1: procedure TRoSExec($M$)
2: Initialize the current configuration by default nodes;
3: Initialize input set $I$;
4: Initialize clock ;
5: while true do
6: Read external input events into $I$;
7: while $I$ is not empty do
8: Execute instantaneous events ;
9: end while
10: Execute timed actions ;
11: Increase clock ;
12: end while
13: end procedure
Transformation of SC into TROS

Diagram:
- GateOpened
- GateClosed
- TrainApproaching
- CloseGate
- TrainMovedOut
- OpenGate
Annotation

TrainApproaching
/ClosedGate

\{(cpu,1), (gate,1)\}_{20}

GateOpened

TrainMovedOut
/OpenGate

\{(cpu,1), (gate,1)\}_{25}

GateClosed
Transformation

TrainApproaching

GateOpened

GateClosing \{ \langle \text{cpu}, 1 \rangle, \langle \text{gate}, 1 \rangle \}^{20}

GateClosed

GateOpening \{ \langle \text{cpu}, 1 \rangle, \langle \text{gate}, 1 \rangle \}^{25}

TrainMovedOut

/CloseGate

/OpenGate
Algorithm 1 TRoSexec

1: procedure TRoSexec($M$)
2:     Initialize the current configuration by default nodes;
3:     Initialize input set $I$;
4:     Initialize clock ;
5:     while true do
6:         Read external input events into $I$;
7:         while $I$ is not empty do
8:             Execute instantaneous events ;
9:         end while
10:     Execute timed actions ;
11:     Increase clock ;
12:     end while
13: end procedure
ACSR

- A formal specification for real-time systems
- The actions of ACSR consists of two kinds of actions: timed actions and event actions.
Semantics

The non-prioritized relation of ACSR
there are three cases for $\alpha < \beta$:

1) $\alpha$ and $\beta$ are events with the same label and $\beta$ has a higher priority;
2) $\alpha$ and $\beta$ are actions and $\beta$ uses a subset of resources with the following two conditions: all resources in $\beta$ have at least the same priority as in $\beta$ and at least one at a higher priority than in $\alpha$, and every resources in $\alpha$ that is not in $\beta$ must have a zero priority;
3) $\beta$ is a $\tau$ event with a non-zero priority while $\alpha$ is a timed action;
Contents

- Introduction
- Background
  - TRoIS and ACSR
- TRoS || ACSR for Embedded Software Components
  - Specifying Real-time Embedded Applications in TRoS || ACSR
- Conclusions
TRoS || ACSR

- it represents a composition of two behavioral systems in TRoS and ACSR respectively.
An Simple Example

SEM1_MANAGEMENT

SEM1_V = {SEM1_V + (GET_SEM1,1).SEM1_P + (POST_SEM1,1).SEM1_V ;
SEM1_P = {SEM1_P+ (POST_SEM1,1).SEM1_V ;
Semantics

• The behavior of TRoS || ACSR is defined in the same way defining ACSR,
• First, the non-priority relations of their composite behavior are defined,
• Then, the non-prioritized behaviors are prioritized based on preemption relations
Synchronization Event for TRoS

\[ E ::= \varepsilon \mid (e?, \ ve).E \mid (e!, \ ve).E \]
A TRoS is defined as a tuple \((N,T,E,G,A)\), where

- \(N\) is a set of nodes,
- \(T\) is a set of transitions,
- \(E\) is a set of primitive event expressions,
- \(G\) is a set of guard expressions,
- \(A\) is a set of action expressions.
TRoS in TRoS || ACSR

- $I_B = E \cup A$: Broadcasting event communicating within TRoS
- $I_S$: Synchronization event between TRoS and ACSR
A Behavior of TRoS

- A behavior of a system in TRoS is defined as a set of possible runs [HN96]. A run is a series of statuses. A status is defined by the tuple $s^T = (C_B, C_T, clk, I)$, where
  - $C_B$ denotes a basic configuration,
  - $C_T$ denotes a timed configuration,
  - $clk$ denotes the current time,
  - $I = I_B \cup I_S$. 
A Run of TRoS

• A run of the system is a sequence $c^T = s^T_0 s^T_1 ... s^T_n$, and we let $S^T$ be the set of statuses.
A system of ACSR is represented in labeled transition system $T^A = (S^A, \text{Act}, \rightarrow_{\pi}, s^A_0)$ [LPS07], where:

- $S^A$ is a set of states,
- $\text{Act}$ includes event and timed actions,
- $\rightarrow_{\pi}$ : prioritized transitions,
- $s^A_0$ : Initial parallel processes (A configuration of processes)
• A system of TRoS || ACSR is defined in a labeled transition system $T = (S, A_{sync}, \rightarrow_\sigma, s_0)$, where
  
  $S = \{(s^T, s^A) \mid s^T \in S^T \text{ and } s^A \in S^A\}$,

  $A_{sync} = I_s \cup \{e \mid inv(e) \in Act\} \cup \{\varepsilon\}$,

  $\rightarrow_\sigma :$ prioritized transition relation of TRoS || ACSR

  $s_0 :$ Initial configuration of TRoS || ACSR consisting of default state nodes and initial processes.
The event $\varepsilon$ implies that no action takes place in ACSR when a step in TRoS is executed.

A behavior of TRoS $\parallel$ ACSR is defined by a sequence of $c = s_0\beta_1s_1...\beta_ns_n$, such that $s_i \in S$, $\beta_i \in Act \cup \{\varepsilon\}$, and $(s_i,\beta_i+1,s_i+1) \in \rightarrow \sigma$, for all $0 \leq i < n$. 
Non-prioritized Relation of TRoS || ACSR

\[ s^T_i = (C_B, C_T, clk, I), s^A_i \in S^A \]

**TauAct:**

\[ s^A_i \xrightarrow{\tau, m} s^A_{i+1} \]

\[ (s^T_i, s^A_i) \xrightarrow{\tau, m} (s^T_{i+1}, s^A_{i+1}) \]

**ACSREventAct:**

\[ s^A_i \xrightarrow{e} s^A_{i+1} \]

\[ (s^T_i, s^A_i) \xrightarrow{e} (s^T_{i+1}, s^A_{i+1}) \]

**TRoSEventAct:**

\[ s^T_i \xrightarrow{\gamma} s^T_{i+1}, s^T_{i+1} = (C'_B, C_T, clk, I') \]

\[ (s^T_i, s^A_i) \xrightarrow{\gamma} (s^T_{i+1}, s^A_{i+1}) \]

, where \( inv(e) \notin I_S \)

**EventComm:**

\[ s^T_i \xrightarrow{\gamma} s^T_{i+1}, s^A_i \xrightarrow{(a, m)} s^A_{i+1} \]

\[ (s^T_i, s^A_i) \xrightarrow{\gamma, \tau, n+m} (s^T_{i+1}, s^A_{i+1}) \]

, where \( (a, n) \in I_S \) and \( (a, n) \notin I' \)

**TimedAct:**

\[ s^T_i \xrightarrow{A_1} s^T_{i+1}, s^A_i \xrightarrow{A_2} s^A_{i+1} \]

\[ (s^T_i, s^A_i) \xrightarrow{A_1 \cup A_2} (s^T_{i+1}, s^A_{i+1}) \]

, where \( \rho(A_1) \cap \rho(A_2) = \emptyset \)
A Status of TRoS || ACSR

\[ s^T_i = (C_B, C_T, clk, I), \quad s^A_i \in S^A \]
Rule : TauAct

TRoS

- **Wait**
  - Wait for SS
  - System service provided

- **Ready**
  - Interrupt or system call

- **Run**
  - System service provided

ACSR

- **Ready**
  - Interrupt or system call
  - SS is over/return

- **Run**
  - SS is over/return

Diagram:

\[ S_i^A \xrightarrow{(\tau, m)} \pi S_{i+1}^A \]

\[ (S_i^T, S_i^A) \xrightarrow{(\tau, m)} (S_{i+1}^T, S_{i+1}^A) \]
**Rule : TauAct**

**TRoS**
- Wait
- Ready
- Run
- Interrupt or system call
- Wait for SS

**ACSR**
- Ready
- Run
- SS is over/return
- Interrupt or system call

Mathematical expressions:

\[
\begin{align*}
    s_i^A &\xrightarrow{\tau, m} \pi s_i^{A_1} \\
    (s_i^T, s_i^A) &\xrightarrow{\tau, m} (s_i^{T_1}, s_i^{A_1})
\end{align*}
\]
Rule: ACSREventAct

\[
\begin{align*}
    s^A_{i+1} &\xrightarrow{e} s^A_{i+1} \\
    (s^T_i, s^A_i) &\xrightarrow{e} (s^T_i, s^A_{i+1})
\end{align*}
\]

where \( \text{inv}(e) \notin I_S \)
Rule : TRoSSEventAct

\[
\begin{align*}
&\text{TRoS} \\
&s^T_i \rightarrow \gamma \ s^T_{i+1}, s^T_{i+1} = (C'_B, C_T, clk, I') \\
&(s^T_i, s^A_i) \xrightarrow{\epsilon} (s^T_{i+1}, s^A_{i+1})
\end{align*}
\]

\[\text{, where } I_S = \emptyset\]
Rule: TimedAction

\[
\begin{align*}
{s_i^T} &\xrightarrow{A_1} \gamma {s_{i+1}^T}, {s_i^A} &\xrightarrow{A_2} \pi {s_{i+1}^A}, {s_{i+1}^T} = (C_B, C_T, clk + 1, I') \\
(s_i^T, s_i^A) &\xrightarrow{A_1 \cup A_2} (s_{i+1}^T, s_{i+1}^A)
\end{align*}
\]

, where \(\rho(A_1) \cap \rho(A_1) = \emptyset\)
Rule: EventComm

\[ s^T_i \rightarrow_\gamma s^T_{i+1}, s^A_i \xrightarrow{(a!, m)} \pi s^A_{i+1}, s^T_{i+1} = (C'_B, C_T, clk, I') \]

\[ (s^T_i, s^A_i) \xrightarrow{(\tau, n+m)} (s^T_{i+1}, s^A_{i+1}) \]

where \((a?, n) \in I_S\) and \((a?, n) \notin I'\)
Preemption Relation

- **Definition (Preemption Relation for TRoS || ACSR)** Given two actions $\alpha$ and $\beta$, we say that $\beta$ preempts $\alpha$, denoted by $(\alpha < \beta)$, if one of the following cases hold:

1. Both $\alpha$ and $\beta$ are events in $D_e$, where $\alpha = (a,p), \beta = (a,p')$, and $p < p'$
2. Both $\alpha$ and $\beta$ are actions in $D_r$, where
   \[
   (p(\beta) \subseteq p(\alpha)) \land \\
   (\forall (r,p) \in \alpha : ((r,p') \in \beta \Rightarrow p \leq p') \land ((r,p') \not\in \beta \Rightarrow p = 0)) \land \\
   (\exists (r,p') \in \beta : \exists (r,p) \in \alpha : p < p')
   \]
3. $\alpha \in D_R$ and $\beta \in D_E$, with $\beta = (\tau, p)$ and $p > 0$
4. $\alpha = \epsilon$ and $\beta \in D_E$, with $\beta = (\tau, p)$ and $p > 0$
5. $\alpha \in D_R$ and $\beta = \epsilon$. 
Prioritized Transition System

Definition) The labeled transition “$\rightarrow \sigma$” is defined as follows: $s \xrightarrow{\alpha} \sigma s'$ if and only if
1) $s \rightarrow s'$ is an unprioritized transition and
2) there is no unprioritized transition system $s \xrightarrow{\beta} s''$ such that $\alpha < \beta$. 
Contents

• Introduction
• Background
  • TRoIS and ACSR
  • TRoS || ACSR for Embedded Software Components
• Specifying Real-time Embedded Applications in TRoS || ACSR
• Conclusions
Specifying RTOS Services

- Scheduling
- Synchronization
- Communication
- Time management
Semaphore in ACSR

\[
\begin{align*}
\text{SEM\_Pend} &= \{\text{}\}:\text{SEM\_Pend} \\
&\quad + (\text{OSSemPend,1}).\text{SEM\_Post} + (\text{OSSemPost,1}).\text{SEM\_Pend}; \\
\text{SEM\_Post} &= \{\text{}\}:\text{SEM\_Post} + (\text{OSSemPost,1}).\text{SEM\_Pend};
\end{align*}
\]
Message Queue

MngMQ1 = {}:MqMng_MQ_1 + (CreateMQ1,1).MQPend;
MQPend = {}:MQPend + (PostMQ1,1).MQPost1
         + (DeleteMQ1,1).MngMQ1;
MQPost1 = {}:MQPost1 + (PostMQ1,1).MQPost2
         + (PendMQ1,1).MQPend + (DeleteMQ1,1).MngMQ1;
MQPost2 = {}:MQPost2 + (PostMQ1,1).MQPost3
         + (PendMQ1,1).MQPost1 + (DeleteMQ1,1).MngMQ1;
MQPost3 = {}:MQPost3 + (PendMQ1,1).MQPost2
         + (OSMqDelete_MQ_1,1).MngMQ1;
TimerSEC10 = {} : TimerSEC
  + (OSTimeDlySEC10Req, 1).TimerSEC10Ack;
TimerSEC10Ack = {} ^ 10 : TimerSECAckExe;
TimerSEC10AckExe = {} : TimerSECAckExe
  + (OSTimeDlySEC10Ack, 1).TimerSEC10;
Example of TRoS || ACSR
A TRoS || ACSR Model

T1

P1

tr1: createT1

tr2: Ev1

P2: ((cpu,1))

tr3: /OP1Gen(data1)

tr4: (OSMMPutMB1!, 1)
    /buf1 = data1

P3

T2

Q1

tr5: createT2

tr6: Ev2

Q2: ((cpu,2))

tr7: /OP2Gen(data2)

tr8: (OSMMPutMB1!, 2)
    /buf1 = data2

Q3

T3

R1

tr9: createT3

tr10: Ev3

R2

tr11: (OSMBGetMB1!, 3)
    /data = buf1

R3: ((cpu,3))

tr12: /OP3(data1)

OSMBox = MboxMB1;

MboxMB1 = {} : MboxMB1 + (OSMBCreateMQ1, 1), MB1Put ;
MB1Put = {} : MB1Put + (OSMMPutMB1?, 1), MB1Get + (OSMBDeleteMB1?, 1), MboxMB1 ;
MB1Get = {} : MB1Post + (OSMBGetMB1?, 1), MB1Put + (OSMBDeleteMB1?, 1), MboxMB1 ;
A TRoS || ACSR Model

Current Time: 0
A TRoS || ACSR Model

Current Time: 0
A TRoS || ACSR Model

Current Time: 0
A TRoS || ACSR Model

Current Time: 0
A TRoS || ACSR Model
A TRoS || ACSR Model

Current Time: 0
A TRoS || ACSR Model

Current Time: 0
A TRoS || ACSR Model

Current Time: 0
A TRoS || ACSR Model
A TRoS || ACSR Model

Current Time: 0
Contents

• Introduction
• Background
  • TRoIS and ACSR
  • TRoS || ACSR for Embedded Software Components
  • Specifying Real-time Embedded Applications in TRoS || ACSR
• Conclusions
Conclusions

- We define here
  - the semantics of a composition of TRoS || ACSR representing application software and RTOS respectively
Conclusions

• TRoS
  • extends Statecharts in terms of time and resource constraints by annotation methods.
  • presents a way to gain a timed and resource-constrained behavioral model from Statecharts in easy way.
Conclusions

- **ACSR**
  - useful to capture RTOS in easy way
  - provides explicit notion of timely prioritized use of resource
  - able to verified with FM verification tools
Conclusions

- TRoS || ACSR

- defines a composition of two different systems representing application software and platform software, i.e., RTOS.
Conclusions

- This work contributes to
  - independently designing application software and platform software with their appropriate formal specification languages,
  - analyzing their composite behaviors based on our behavioral semantics of TRoS || ACSR.
References


Thanks
Case Study
Avionics Systems based on ARINC 653
ARINC 653
Semaphore in ACSR

BUFFER_SERVICE = SERVICE_BUFF1;

SERVICE_BUFF1 = SERVICE_BUFF_buff1 || STAT_BUFF_buff1;
STAT_BUFF_buff1 = {}
+ (Get_buff1_id,1)
  .(‘get_buff1_id_NO_ERROR,1)
  .STAT_BUFF_buff1
+ (buff1_name_not_identified,1)
  .(‘get_buff1_id_INVALID_CONFIG,1)
  .STAT_BUFF_buff1
+ (Get_buff1_status,1)
  .(‘get_buff1_status_NO_ERROR,1)
  .STAT_BUFF_buff1
+ (buff1_ID_not_identified,1)
  .(‘get_buff1_status_INVALID_CONFIG,1)
  .STAT_BUFF_buff1;

SERVICE_BUFF_buff1 = CREATEBUFF_buff1;

CREATEBUFF_buff1 = {}
+ (Crt_buff1,1)
.CrtBUFFRtnCdbuff1;
CrtBUFFRtnCdbuff1 = (‘crtbuff1_NO_ERROR,1)
.BUFFinSVC_buff1
+ (no_enough_buff_space,1)
Verification

State machine contains 153 reachable states (0 deadlocked), 153 edges.
Edges represent 120 timed transitions,
    27 internal actions, and
    6 external untimed actions.
LTS is non-Zeno.
6 non-deadlocked states are capable of stopping the clock.
Time to compute LIS: 0 seconds user time; 0 seconds system time.