Formally Specifying and Verifying Real-Time Systems with ASTRAL

Richard A. Kemmerer

Reliable Software Group
Computer Science Department
University of California, Santa Barbara

What is a Real-Time System?

- A system whose semantics depend on the speed of execution of (some of) the activities
- A system where a failure to produce certain results within given time limits (too early ... too late) may result in an error (whose effect may be catastrophic)
- A system whose performance and correctness can not be separated

Real-Time Systems

- In 1977 Wirth classified programs into three types
  - Sequential
  - Parallel
  - Processing-time dependent (i.e., real-time)

Verifying Real-Time Systems

- Sequential and real-time systems both have critical functionality requirements
- Real-time systems must also meet critical performance deadlines

ASTRAL Solution

- Develop a formal specification language
- Develop a formal proof system for proving properties about the specifications
- Build tools to support the construction and use of the specifications

Goals for ASTRAL

- Language usability was a major design factor
- Tool development proceeded in parallel with the language development
- Specifications are layered, compositional, and executable
Layered, Compositional, and Executable Specifications

• Specification modules are refined to include more detail without changing their interface
• Behavior of the whole is determined by the behavior of the parts
• Allow the developers to treat the specifications as prototypes

An Overview of ASTRAL

• In ASTRAL a real-time system is modeled by a collection of process type specifications and a single global specification
• The global specification contains declarations for types, constants, etc. that are shared among process types
• A process type specification contains types, state variables, transitions, etc.
• Every process is thought as being in various states, with one state differentiated from another by the values of state variables
• Only state transitions can change the values of state variables; Transitions are described in terms of pre- and post-conditions by using an extension of first-order predicate calculus

The ASTRAL Computational Model

• Maximal parallelism among processes
• Non interruptable, non overlapping transitions in a single process instance
• Transitions are executed as soon as they are enabled, that is, their pre-condition is satisfied (exception: exported transitions)
• Implicit one-to-many message passing communication
• Time can be continuous or discrete

The ASTRAL Computational Model

• Every process can export state variables and transitions
• Inter-process communication is accomplished by inquiring about the value of exported variables and the start time and end time of exported transitions.
  – i.Start(Op, t) true iff the last occurrence of transition Op of instance i started at time t.
  – i.End(Op, t) true iff the last completed occurrence of transition Op of instance i ended at time t.
  – past(expr, t) represents the value of expr at time t

Environmental Assumptions

• An environment clause formalizes the assumptions that must always hold on the behavior of the external environment
• For each process there is a local environment clause which expresses the assumptions about calls to the exported transitions
• There is also a global environment clause which is a formula that may refer to all exported transitions in the system
Environmental Assumptions

- An exported transition can fire only after it has been called by the environment
- If Op is an exported transition, Call(Op, t) is true iff at time t the last occurrence of the call to Op occurred

System Assumptions

- Each process p may have an imported variable clause which formalizes assumptions that process p makes about the context provided by the other processes in the system

Critical Requirements

Critical requirements are expressed by means of:
- invariants (global and local)
- schedules (global and local)

Invariants

- Must be true regardless of the environment or the context in which the process or system is running
- State properties that must initially be true and must be guaranteed during system evolution

Schedules

- Schedules are additional system properties that are required to hold under more restrictive hypothesis than invariants
- Assumptions expressed in the associated environment, imported variable clauses and/or system assumptions may be used to prove the validity of a schedule

Further Assumptions and Restrictions Clause

- Used to prove that schedules are feasible
- Serve only as a guidance to the implementer
- Further environment assumptions
- Further process assumptions (FPAp) section restricts the possible system implementations and reduces the level of nondeterminism of the system specification
  - transition selection part
  - constant refinement part
Telephony Example

- Two process type specifications
  - Phone
  - Central Control
- One instance of central control
- One instance of phone for each operating telephone number in the area

Keywords

Now Represents the current value of time
Self Is used by a process when it wants to refer to its own id

Global

**PROCESSES**
- Phones: array[1 .. Num_Phone] of Phone,
- Central: Central_Control

**TYPE**
- Positive_Integer i: Integer (i > 0),
- Digit IS TYPEDEF d: Integer (d ≥ 0 & d ≤ 9),
- Digit_List IS LIST OF Digit,
- Phone_ID IS TYPEDEF pid: ID
  (IDTYPE(pid)=Phone)
- Enabled_State = (Idle, Ready_To_Dial, Dialing, Ringing, Waiting, Talk, Disconnect, Busy, Alarm)

**CONSTANT**
- Num_Phone, Max_Cust : Positive_Integer

Specification Function

IDTYPE Returns the process type when presented with a process id
Phone

IMPORT
Digit, Phone_ID, Enabled_State,
Central.Phone_State,
Central.Enabled_Ring_Pulse,
Central.Enabled_Ringback_Pulse

EXPORT
Offhook, Next_Digit, Pickup, Enter_Digit,
Hangup

Phone

VARIABLE
Offhook, Dialtone, Ring, Ringback,
Busytone: Boolean,
Next_Digit: Digit

Phone

TRANSITION Pickup T1
ENTRY
~Offhook
EXIT
Offhook
& ~Dialtone
& ~Busytone
& ~Ring
& ~Ringback

Phone

TRANSITION Start_Tone T2
ENTRY
Offhook & ~Dialtone
& Central.Phone_State(Self)=Ready_To_Dial
& FORALL t:Time(
Change(Dialtone,t) →
t < Change(Offhook) )
EXIT
Dialtone

Phone

TRANSITION Enter_Digit(D:Digit) T4
ENTRY
Offhook
& ( Central.Phone_State(Self)=Ready_To_Dial & Dialtone
| Central.Phone_State(Self)=Dialing)
EXIT
Next_Digit=D & ~Dialtone

Other Phone Transitions

Start_Ring
Stop_Ring
Start_Ringback
Stop_Ringback
Start_Busytone
Stop_Busytone
Hangup
Phone Environment

ENVIRONMENT
FORALL t: Time (Call(Pickup, t) → ~past(Offhook, t))
& FORALL t: Time (Call(Hangup, t) → past(Offhook, t))
& FORALL t: Time (Call(Pickup, t) → Call(Pickup) - Call(Pickup) ≥ 1)

Phone Schedule

SCHEDULE
FORALL t: Time (Call(Pickup, t)  \rightarrow  Start(Pickup, t))

Phone Invariant

INVARIANT
(Dialtone → ~Ring & ~Ringback & ~Busytone)
& (Ring → ~Dialtone & ~Ringback & ~Busytone)
& (Ringback → ~Dialtone & ~Ring & ~Busytone)
& (Busytone → ~Dialtone & ~Ring & ~Ringback)

Phone Further Assumption

FURTHER ASSUMPTION #1
FURTHER PROCESS ASSUMPTION
TRANSITION SELECTION
enabled_transitions CONTAINS
any_subset({Stop_Ringback, Stop_Busytone})
& TRUE
→ eligible_transitions =
{Stop_Ringback, Stop_Busytone}
INTERSECT enabled_transitions

Central Control

IMPORT
Digit, Digit_List, Phone_ID, Enabled_State, Phones.Offhook, Phones.Next_Digit, Phones.Pickup, Phones.Enter_Digit

EXPORT
Phone_State, Enabled_Ring_Pulse, Enabled_Ringback_Pulse
Central Control

VARIABLE
Phone_State(Phone_ID): Enabled_State,
Enabled_Ring_Pulse(Phone_ID):Boolean,
Enabled_Ringback_Pulse(Phone_ID): Boolean,
Connected_To(Phone_ID): Phone_ID,
Number(Phone_ID): Digit_List

Central Control

IMPORTED VARIABLE CLAUSE

SETSIZE( { SETDEF P: Phone_ID (Now - 2 <= P.Start(Pickup) <= Now) } ) <= Max_Cust

Central Control

TRANSITION
Give_Dial_Tone(P:Phone_ID) Tim1
ENTRY
P.Offhook
& Phone_State(P)=Idle
EXIT
Phone_State(P) BECOMES Ready_To_Dial
& Number(P) BECOMES NIL

Central Control

TRANSITION
Process_Digit(P:Phone_ID) Tim2
ENTRY
P.Offhook
& Count(P) < 7
& ( ( Phone_State(P)=Ready_To_Dial
& P.End(Enter_Digit) > End(Give_Dial_Tone(P))
| ( Phone_State(P)=Dialing)
& P.End(Enter_Digit) > End(Process_Digit(P)))
EXIT
Number(P) BECOMES Number'(P) CONCAT LISTDEF(P.Next_Digit)
& Phone_State(P) BECOMES Dialing

Central Control

TRANSITION
Process_Call
Enable_Ring_Pulse
Disable_Ring_Pulse
Enable_Ringback_Pulse
Disable_Ringback_Pulse
Start_Talk
Terminate_Call
Generate_Alarm

Central Control

INVARIANT
FORALL P:Phone_ID (Count(P) >= 0 & Count(P) <= 7
& Phone_State(P)=Waiting → Phone_State(Connected(P))=Ringing
& Phone_State(P)=Ringing → Phone_State(Connected(P))=Waiting
& Phone_State(P)=Talk →
Central Control Constraint

CONSTRAINT
FORALL P:Phone_ID (
   | Phone_State(P)=Busy
   | Phone_State(P)=Alarm
   | Phone_State(P)=Disconnect
   & Phone_State(P) ~ Phone_State'(P)
   → Phone_State(P)=Idle )

Central Control Schedule

SCHEDULE
FORALL P:Phone_ID ( Phone_State(P)=Ringing
   & Now - End(Process_Call(Connected(P))) >= Downtime_Ring
   → EXISTS n:Integer ( Endn(Enable_Ring_Pulse(P)) >
   End(Process_Call(Connected(P))) & Endn(Enable_Ring_Pulse(P)) <=
   End(Process_Call(Connected(P))) + Downtime_Ring )
)

Global Environment

ENVIRONMENT
SETSIZE( SETDEF P: Phone_ID ( Now - 2 <= P.Call(Pickup) <= Now) ) <= Max_Cust

Global Schedule

SCHEDULE
FORALL P:Phone_ID, t: Time ( P.Offhook & P.Call(Pickup, t)
   & Now - t >= 2 & past(Central.Phone_State(P), t) = Idle
   → EXISTS t1:Time ( t < t1 <= t + 2 & past(Central.Phone_State(P), t1)
   = Ready_To_Dial) )

Formal Proofs

Formal proofs in ASTRAL can be divided into two categories:
- Intra-level proofs
- Inter-level proofs

Formal Proofs

- Intra-level proofs deal with proving that the specification of level i is consistent and satisfies the stated critical requirements

- Inter-level proofs deal with proving that the specification of level i+1 is consistent with the specification of level i
ASTRAL Intra-level Proofs

- Every process specification guarantees its local invariant
- Every process specification guarantees its local schedule
- The specification guarantees the global invariant
- The specification guarantees the global schedule
- The imported variable assumptions are guaranteed by the specification
- All the assumptions about the environment are compatible

Building an ASTRAL intra-level proof

- The local invariant $I_p$ describes properties which are independent from the environment, i.e., it must hold in every possible environment
- To prove that the process $P_p$ guarantees $I_p$ we have to show that:
  1. $I_p$ holds in the initial state of process $p$, and
  2. If $P_p$ is in a state in which $I_p$ holds, then for every possible evolution of $P_p$, $I_p$ will hold.

Building an ASTRAL invariant proof

1. $\text{Init\_State}_p \land \text{Now} = 0 \implies I_p$
2. We assume that $I_p$ holds until a given time $t_0$ and prove that $I_p$ will hold for every time $t > t_0$. Without loss of generality we assume that $t = t_0 + \Delta$, for some fixed $\Delta > 0$ and we will show that $I_p$ holds until $t_0 + \Delta$.

   We may need to make assumptions on the possible sequences of events that occurred within the interval $[t_0 - H, t_0 + \Delta]$, where $H$ is a constant a priori unbounded. By event we mean the starting (ending) of some transition of $P_p$.

ASTRAL Abstract Machine Semantics

Captured in three axioms

A1 start to end of transition is equal to the transition duration
A2 if processor is idle and some transitions are enabled, then one will fire
A3 for each processor the transitions are nonoverlapping

Axiom A1

$$\forall t : \text{Time}, \text{Op} : \text{Trans\_of\_p} \left( \text{Now} - t \geq T\text{Op} \rightarrow \left( \text{past}(\text{Start}(\text{Op}), t) = t \leftrightarrow \text{past}(\text{End}(\text{Op}), t + T\text{Op}) = t + T\text{Op} \right) \right)$$

where $T\text{Op}$ represents the duration of $\text{Op}$

Axiom A2

$$\forall t : \text{Time} \left( \exists d : \text{Time}, ST : \text{SET OF Trans\_of\_p}(\text{t}) \right)$$

FORALL $t' : \text{Time}, \text{Op} : \text{Trans\_of\_p}$

- $t \geq t' - d$ & $t' < t$ & $\text{Op}$ ISIN ST
- & \text{past}(\text{Start}(\text{Op}), t') < \text{past}(\text{End}(\text{Op}), t)$$
- & $ST \subseteq ST$ & $ST \rightarrow \text{EMPTY}$
- & $\forall \text{Op}' : \text{Trans\_of\_p}$
- & $\text{Op}'$ ISIN ST $\rightarrow \text{Eval\_Entry}(\text{Op}', t')$
- & $\forall \text{Op}' : \text{Trans\_of\_p}$
- & $\text{Op}'$ ISIN ST $\rightarrow \rightarrow \text{Eval\_Entry}(\text{Op}', t')$
- & $\rightarrow \text{UNIQUE} \text{Op}' : \text{Trans\_of\_p}$
- & $\text{Op}'$ ISIN ST & \text{past}(\text{Start}(\text{Op}', t), t)=$(t))$
Axiom A3

FORALL t1, t2:Time, Op: Trans_of_p(
   Start(Op)=t1 & End(Op)=t2 & t1 < t2
→ FORALL t3:Time, Op': Trans_of_p(
   t3 ≥ t1 & t3 < t2 & Start(Op')= t3
   → Op = Op' & t3 = t1)
& FORALL t3: Time, Op': Trans_of_p(
   t3 > t1 & t3 ≤ t2 & End(Op')= t3
   → Op = Op' & t3 = t2))

Building an ASTRAL invariant proof

• For each sequence of events σ a formula Fσ can be algorithmically associated with σ

• For each event occurring at time t we have:
  - past(ENpj, t) & past(Start(Oppj, t), t) if the event is the start of Oppj
  - past(EXpj, t) & past(End(Oppj, t), t) if the event is the end of Oppj

A1 & A2 & A3 ⊢ Fσ & FORALL t:Time (t < t0 → past(Ip,t))
   → FORALL t1:Time (t1 > t0 & t1 <= t0 + Δ → past(Ip, t1))

To Prove the Schedule

Need to modify axiom A2 to deal with the transition selection clause and to require calls for external transitions

Also need an axiom to state that a call is issued only if a call was made from the environment and not yet serviced

Axiom A2'

FORALL t:Time (t
   EXISTS d: Time, ST: SET OF Trans_of_p(
      FORALL t1:Time, Op: Trans_of_p(
         t1 ≥ t - d & t1 < t & Op ISIN ST
      & past(Start(Op),t1) < past(End(Op),t)
      & ST ⊆ ST & ST = EMPTY
      & FORALL Op':Trans_of_p (Op' ISIN ST → Eval_Entry(Op',t))
      & FORALL Op':Trans_of_p (Op' ~ISIN ST → ~Eval_Entry(Op',t))
      → UNIQUE Op':Trans_of_p (Op' ISIN TS(ST) & past(Start(Op'),t1)))
   & EXISTS t1: Time(t1 ≤ Now & Call(Op, t1)
   & FORALL t: Time (t ≥ t1 & t ≤ Now & ~Start(Op,t)
   → past(Issued_call(Op),t)))
   & EXISTS t1: Time(t1 ≤ Now & Start(Op, t1)
   & FORALL t: Time(t > t1 & t ≤ Now & ~Call(Op,t)
   → ~past(Issued_call(Op,t))))

Building an ASTRAL schedule proof

• To prove that Pp guarantees the local schedule Scp:
  1) Scp holds in the initial state of process p, and
  2) If Pp is in a state in which Scp holds, then for every possible evolution of Pp compatible with the system assumptions, when the environment behaves as described in Pp, Scp will hold.

• We can assume that the local invariant Ip holds
Building an ASTRAL Global proof

• Proving that S guarantees the global invariant IG is done as for the local invariant case:
  1) IG holds in the initial state of S, and
  2) If S is in a state in which IG holds, then for every possible evolution of S, IG will hold.

• We can assume that the local invariants Ip hold

Building an ASTRAL global proof

• To prove that S guarantees the global schedule ScG:
  1) ScG holds in the initial state of S, and
  2) If S is in a state in which ScG holds, then for every possible evolution of S, ScG will hold.

• We can assume that the global invariant, every local invariant and schedules, and the global environment assumptions hold

• We cannot use any of the local environment assumption or system assumptions to prove the validity of the global schedule

Building an ASTRAL consistency Proof

• When proving a local schedule we rely on the assumptions on the imported variables. Such assumptions must be checked against the behavior of the processes they are imported from.

• Every process contains two clauses describing assumptions on the behavior of the environment. The global specification contains another clause describing assumptions on the environment. We must verify that all the assumptions do not contradict each other

ASTRAL Inter-level Proofs

• Every transition at level n is correctly implemented at level n+1

Composing ASTRAL Specifications

Composing two top level specifications S’ and S” means to build a new top level specification C, that is the specification of a system obtained by making one or more instances of S’ and S” interact

In order to compose S’ and S” one has to define:
- how the interaction between S’ and S” can be formally defined
- how the specification C can be built starting from S’, S” and the description of their interaction
- under which conditions the critical requirements of S’ and S” will still be valid in C.

The Compose Section

• The COMPOSE section allows ASTRAL specifications to be composed into a new specification of a more complex system
• By adding the COMPOSE section and introducing a compositional specification method a system designer can now reason about the behavior of a composite system in terms of its components
• The size of the composite specification grows linearly with the size of the component specifications
The Compose Section

- The COMPOSE Section describes the interaction between $S'$ and $S''$
- Some exported transitions of $S'$ ($S''$) are no longer exported, i.e., the stimuli needed to fire such transitions are produced by $S''$ ($S'$) rather than the external environment

\begin{align*}
\text{FORALL } t: \text{Time}, \ldots \ (P(S') \leftrightarrow \text{Call}(T, t))
\end{align*}

where $P(S')$ is an ASTRAL well-formed formula describing the occurrence of events in $S'$ equivalent to calling transition $T$ of $S''$.

Building the New Specification

- When composing two specifications by means of a COMPOSE section it is desirable to automatically produce the specification of the composed system. Therefore it is necessary to:
  - Build the global specification
  - Modify the process specifications

The Global Specification

- The clauses defining types, constants, and definitions are taken from the COMPOSE section and the global specifications of $S'$ and $S''$ (using the name clash clause)
- The global invariant (schedule) is built from the invariants (schedules) of $S'$ and $S''$, by substituting any occurrences of the operators $\text{Start}$, $\text{End}$ and $\text{Call}$ referring to a no longer exported transition with an equivalent predicate referring to exported variables
- The global environment clause is similarly built by modifying those parts that refer to no longer exported transitions.

The Process Specifications

- All process specifications belonging to either $S'$ or $S''$ belong to $C$
- For each process specification the following modifications are required:
  - Local environment clause: Remove references to no longer exported
  - Export/Import clause: Unexport transitions referred to in the Call Generation Clause of the COMPOSE section
  - Import variables referred to in the CGC
  - Transitions in the CGC: Modify the Entry condition:
    \begin{align*}
    \text{EXISTS } t: \text{Time}, \ldots \ (P(S') \land \text{Start}(T) < t) \land \text{Old_Entry},
    \end{align*}
    where $P(S')$ is the predicate used in the CGC

Imported variables clause: Add assumptions about new imported variables

Local Schedule: Modify as for Global Schedule

Local Invariant: No modification is required

Process assumptions: No modification is required
**Proof Obligations**

- Under what conditions are the invariants and schedules of $S'$ and $S''$ still valid in $C$?
- Invariants do not depend on the environment
  Therefore, they are still valid
- Schedules do depend on the environment
  Therefore, one has to prove that the behavior of $S'$ ($S''$) implies what is stated in the environment clauses of $S'$ ($S''$):

$$A_1 & A_2 & A_3 & A_4 & \text{Env} \quad \text{G}_1' & \text{CG}' \quad \text{F}_{\sigma}' \rightarrow \text{Env} \quad \text{G}_2', \text{ for } S'$$

$$A_1 & A_2 & A_3 & A_4 & \text{Env} \quad \text{G}_1'' & \text{CG}'' \quad \text{F}_{\sigma}'' \rightarrow \text{Env} \quad \text{G}_2'', \text{ for } S''$$

**Modularized Specifications**

- The composite specification approach coupled with the top-down refinement specification approach allows a system designer to specify his/her system using either a bottom-up approach or a top-down approach or a combination of the two
- The composability of specifications also promotes the reuse of existing specifications

**ASTRAL Tool Suite**

In order to get designers to use formal methods to develop real-time systems it is necessary to provide them with an integrated set of tools for writing and analyzing their specifications

**ASTRAL Software Development Environment (SDE)**

- Syntax directed editor
- Specification processor
- Reasoning system
- Specification testing component
- Browser kit

**SDE**

- A key design criteria for the SDE was that a user should never need to switch between tools nor should there be any need for data exchange via temporary files
- The user should be able to change from specification writing to type checking to generating proof obligations with a mere button push

**Syntax-Directed Editor**

- Only basic editing functions
- Syntax is automatically checked when input
  - errors are indicated when entered
  - help facility displays corresponding part of the grammar
- When editing of a section is completed the section of text is formatted into a fixed format
**Specification Processor**

- Validation component checks the entire specification for:
  - type errors
  - scoping errors
  - missing parameters

- Proof obligation component generates
  - Intra-level proof obligations
  - Inter-level proof obligations
  - Composition proof obligations

**Reasoning System**

- Uses PVS theorem prover
  - ASTRAL semantics have been written in the PVS specification language
  - ASTRAL to PVS translator

- ASTRAL model checker

**Specification Testing Tool**

- Symbolically executes an ASTRAL specification under user direction

- Modification of the ASTRAL model checker

**Browser Kit**

- Uses three inter-related databases
  - Variables database
  - Transitions database
  - Processes database

- Automatically updated whenever the specification is edited

- Browser for each database

- Particularly useful during the maintenance phase

**ASTRAL Case Studies**

- Standard Benchmarks
  - railroad crossing
  - elevator

- Phone System (POTS)
- Wide-area Phone System

- Hardware
  - checksum generator
  - UART

- Robot Control System

**For More Information**

- Introduction to Language and Composition
  - TSE September 1997

- Intra-level Proof Obligations
  - TSE August 1994

- Inter-level Proof Obligations
  - ESEC 95

- Composability Proof Obligations
  - ISSTA 93
Software Development Environment (SDE)
ftp from ftp.cs.ucsb.edu in directory seclab-distrib

Using the ASTRAL Model Checker Tool for Encryption Protocol Analysis
– DIMACS 97
– FM&SP 98

Online ASTRAL Info
www.cs.ucsb.edu/~seclab/projects/ASTRAL