The Timed Abstract State Machine (TASM) Language and Toolset

Kristina Lundqvist, Martin Ouimet

Embedded Systems Laboratory
Massachusetts Institute of Technology
January 26th, 2007

The TASM Research Team

- Dr. Kristina Lundqvist – Professor and Supervisor
- Martin Ouimet – PhD Student
- David C. Wang – Master’s Student
  - Brittany Baker
  - Guillaume Berteau
  - Mathieu Quenot
Embedded Systems Laboratory (ESL)

- Part of the Aeronautics and Astronautics Department at MIT
  - Founded in 2002
  - Projects include:
    - System on Chip (SoC) Design Approaches
    - Hardware-Software Co-Synthesis
    - Formal Verification of Hardware-Software Systems
    - Real-Time System Specification and Analysis
    - Software Process Improvement

Application Domain

How do you account for technology obsolescence in a rapidly changing environment?

Source: DSB Briefing, Dan Czelusniak, 1998

In operation 30 to 90 years
Ongoing research

Current State

Ongoing Research

History of the TASM Project

- The Goal:

- The Specifics:
  - Based on a formal, yet usable specification language
  - Abilities to specify and reason about functional behavior and non-functional behavior (time, resources)
  - Automate as much as possible (verification, test case generation)
  - Be able to work at different Levels of abstraction
  - End to end traceability
The Production Cell

<table>
<thead>
<tr>
<th>Component</th>
<th>Action</th>
<th>Time</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feed</td>
<td>Move block from beginning to end</td>
<td>5</td>
<td>500</td>
</tr>
<tr>
<td>Deposit</td>
<td>Move block from beginning to end</td>
<td>7</td>
<td>500</td>
</tr>
<tr>
<td>Robot</td>
<td>Rotate 30°</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>Robot</td>
<td>Pick up a block</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>Robot</td>
<td>Drop a block</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>Press</td>
<td>Stamp a block</td>
<td>11</td>
<td>1500</td>
</tr>
</tbody>
</table>
The TASM Tool

- Demo 1

Target Systems - I

- Reactive:
  - Typically do not terminate but are in constant interaction with the environment

- Real-Time:
  - Need to react to environmental stimulus within an acceptably bounded amount of time

- Embedded:
  - Have limited resources in terms of memory, bus bandwidth, etc.
Target Systems - II

- Examples:
  - Avionics
  - Embedded Controllers

History of the TASM Project - II

- The Approach:
  - The Quest for a Specification Language
    - time/timed petri nets,
    - timed automata,
    - timed process algebra,
    - ASM, etc.

- What we liked about ASM:
  - Text-based
  - Literate
  - Not tied to a particular verification method
  - Good history of case studies
  - Theory can be refined to suit a particular purpose
Time Concepts

- Time Concepts in Modeling Languages
  - Time passage
  - Duration

- Time Specification
  - Single value
  - Interval

Incorporating Time into ASM

- “Time” external function:
  - useful to describe time passage
  - a bit awkward to describe duration

- Attaching Durations to ASM:
  - State (duration between 2 steps)
  - Transition (duration of a step)
The TASM Language

- Basic Concepts: ASM +
  - Types
  - Durative steps
  - Resources consumed during steps
  - Hierarchical composition
  - Parallel composition
  - Synchronization primitives

- ASM Subset included in TASM:
  - “Block form” (Gurevich, Gargantini, …)
  - if then
  - Variable assignments (nullary functions only)
  - Similar to subset of Winter (Model Checking for ASMs)

The TASM Language – Abstract Syntax I

- \text{ASMPEC} = \langle E, MS \rangle
  - \text{E} is the Environment \langle TU, VU, RVS, CS \rangle
    - TU is a set of types (Boolean, Integer, Float, UDT)
    - VU is a set of typed variables
    - RVS is a set of resources
    - CS is a set of synchronization channels
  - MS is a set of Machines M = \langle MV, CV, RS \rangle
    - MV is a set of monitored variables for the machine
    - CV is the set of controlled variables for the machine
    - RS is a set of rules R
The TASM Language – Abstract Syntax II

- \( R \in RS = <T, RC, G, E, CS, CR> \)
  - \( T \) is the duration of the rule execution, in the form \([a, b)\) where \(a, b \geq 0\) and \(a \leq b\)
  - \( RC \) is the set of consumed resources \( P \) during the rule execution, where \( P \) is of the form \([l, u)\) where \(l, u, \geq 0\) and \(l \leq u\)
  - \( G \) is the rule guard, a boolean predicate
  - \( E \) is the rule effect, a set of assignments of values to variables

The TASM Language – Abstract Syntax III

- From here on out, we will use
  - \( M_i \) to denote the \( i^{\text{th}} \) Machine of a Specification
    - \( M \) if only one Machine is present
  - \( R_{ij} \) to denote the \( j^{\text{th}} \) Rule of the \( i^{\text{th}} \) Machine
    - \( R_j \) if only one Machine is present
  - Analogously, \( G_{ij}, E_{ij} \)

- We will use the term “Environment” informally
The TASM Language – Concrete Syntax I

- Textual format (CFG)
- XML format
- Tool format:

```text
//This rule flips the state of light1 and switch1 from OFF, UP to ON, DOWN
RULE1
{
  t := [1, 5];
  memory := [10, 128];
  bus := 345;
  if light1 = OFF and switch1 = UP then
    light1 := ON;
    switch1 := DOWN;
}
```

Time and resource annotations can be expressed using a single value or using an interval

The TASM Language – Semantics I

- A step $k = a$ rule execution
- State $S_k = Time \times Resource\ Values \times Value\ of\ all\ Variables = <g_{tk}, RVS, VARS>$
- Update set $US_k = <t_k, RCS_k, UVS_k>$
  - $t_k$ = duration of rule execution
  - $RCS_k$ = set of consumed resources
  - $UVS_k$ = set of updates to variables
The TASM Language – Semantics II

- A Machine $M$ yields an update set $US_k$ in State $S_k$ if:
  - One of its Rules $R_j$ is “enabled” in State $S_k$:
    - If $G_j$ evaluates to true in State $S_k$
    - $t_k \in T_j$
    - Each resource consumed, $r_{ci} \in RC_j$
    - $UVS_k \subseteq E_j$

- Operator $\circ$: State x Update Set $\rightarrow$ State
  - $S_{k+1} = S_k \circ US_k$
  - $S_{k+1} = <g_{t_k} + t_k, RCS \circ RVS, VARS \circ UVS_k> $

---

The TASM Language – Example I

//This rule flips the state of light1 and switch1 from OFF, UP to ON, DOWN

RULE1
{
  t := [1, 4];
  memory := 10;
  bus := 345;

  if light1 = OFF and switch1 = UP then
    light1 := ON; switch1 := DOWN;
}

//This rule flips the state of light1 and switch1 from OFF, UP to ON, DOWN

RULE2
{
  t := 3;
  memory := [5, 40];
  bus := 450;

  if light1 = ON and switch1 = DOWN then
    light1 := OFF; switch1 := UP;
}
The TASM Language – Example II

• Possible Run:
  - $S_0 = <0, \langle \text{memory, 0} \rangle, \langle \text{bus, 0} \rangle>, \langle \text{light1, OFF} \rangle, \langle \text{switch1, UP} \rangle>$
  - $US_0 = <2, \langle \text{memory, 10} \rangle, \langle \text{bus, 345} \rangle>, \langle \text{light1, ON} \rangle, \langle \text{switch1, DOWN} \rangle>$
  - $S_1 = S_0 \circ US_0 =$
    - $<2, \langle \text{memory, 10} \rangle, \langle \text{bus, 345} \rangle>, \langle \text{light1, ON} \rangle, \langle \text{switch1, DOWN} \rangle>$
  - $US_1 = <3, \langle \text{memory, 32} \rangle, \langle \text{bus, 450} \rangle>, \langle \text{light1, OFF} \rangle, \langle \text{switch1, UP} \rangle>$
  - $S_2 = S_1 \circ US_1 =$
    - $<5, \langle \text{memory, 32} \rangle, \langle \text{bus, 450} \rangle>, \langle \text{light1, OFF} \rangle, \langle \text{switch1, UP} \rangle>$
  - ...

The TASM Language – Static Analysis I

• A Machine $M$ is *Complete* if the disjunction of its Rule Guards $G_j$ $j \geq 0$, $j \leq n$ forms a tautology:
  - $G_0 \lor G_1 \lor \ldots \lor G_n$

• A Machine $M$ is *Consistent* if the disjunction of the pairwise conjunctions of its Rule Guards $G_j$ $j \geq 0$, $j \leq n$ is invalid:
  - $(G_0 \land G_1)$ or $(G_0 \land G_2)$ or $\ldots$ $(G_{n-1} \land G_n)$
The TASM Language – Hierarchical Composition I

- Terms and concepts borrowed from XASM (Anlauff)
  - Function Machine (macro)
  - Sub Machine (procedure)

- A Function Machine FM is similar to Machine M:
  - Has a set of input parameters IP
  - Returns a single value v
  - Has no side-effect (resources, time, environment)
  - Can occur in Rule Guards or in Rule Effect expressions

The TASM Language – Hierarchical Composition II

- Example:

```
FM: whatshoes

R1: Nice out
    { if innice = true then out := sneakers; }

R2: Not Nice out
    { if innice = false then out := boots }

FM: niceout

R1: Nice out
    { if intemperature >= 15 then out := true }

R2: Not Nice out
    { if intemperature < 15 then out := false }

FM: outfit

R1: Nice
    { if niceout(temperature) = true then
      shoes := whatshoes(true)
      shirt := t_shirt
      mood := happy
    }

R2: Not nice
    { if niceout(temperature) = false then
      shoes := whatshoes(false)
      shirt := sweater;
      mood := sade
    }
```
The TASM Language – Hierarchical Composition IV

- Theorem: For every Machine $M$ containing a call to a Function Machine $FM$ in one of its rule guards, there exists an equivalent Machine $M'$ without the call to the Function Machine

- Constructive Proof Idea:
  - “Inline” Function Machine call by replacing expression using $FM$ in $M$
  - One rule with a Function Machine call $FM$ could yield $n$ Rules in $M'$ if the Function Machine $FM$ has $n$ rules

The TASM Language – Hierarchical Composition V

- Example:

<table>
<thead>
<tr>
<th>FM: niceout</th>
<th>M: outfit</th>
<th>M': outfit</th>
</tr>
</thead>
</table>
| R1: Nice out
  { if intemperature $\geq 15$ then out := true } | R1: Nice
  { if niceout(temperature) = true then
    shoes := whatshoes(true);
    shirt := t_shirt;
    mood := happy; }
  R2: Not Nice out
  { if intemperature < 15 then out := false; } | R1: Nice
  { if temperature $\geq 15$ then
    shoes := whatshoes(true);
    shirt := t_shirt;
    mood := happy; }
  R2: Not Nice
  { if temperature < 15 then
    shoes := whatshoes(false);
    shirt := sweater;
    mood := sad; }

K. Lundqvist and M. Quimet – tasm@mit.edu 27 (63)
The TASM Language – Hierarchical Composition VI

- A Sub Machine \( SM \) has the same definition as the definition of a Machine \( M \) given previously
  - The semantics of a Sub Machine are also identical to those of \( M \)
  - Sub Machines can occur only in effect expressions \( E_i \)

---

The TASM Language – Hierarchical Composition VII

- Example:

<table>
<thead>
<tr>
<th>SM: niceoutfit</th>
<th>FM: niceout</th>
<th>M: outfit</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: Nice out</td>
<td>R1: Nice out</td>
<td>R1: Nice</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>if true then</td>
<td>if intemperature &gt;= 15 then</td>
<td>Nice</td>
</tr>
<tr>
<td>shoes := whatshoes(false);</td>
<td>out := true</td>
<td></td>
</tr>
<tr>
<td>shirt := t_shirt;</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>mood := happy;</td>
<td>R2: Not Nice out</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if intemperature &lt; 15 then</td>
<td></td>
</tr>
<tr>
<td></td>
<td>out := false</td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

SM: niceoutfit
FM: niceout
M: outfit
The TASM Language – Hierarchical Composition VIII

Theorem: For every Machine $M$ containing a call to a Sub Machine $SM$ in one of its rule guards, there exists an equivalent Machine $M'$ without the call to the Sub Machine

Constructive Proof:
- Sub Machine $M_0$ has $n$ Rules $R_{01} \ldots R_{0n}$
- Machine Rule $R_{1j}$ of Machine $M_i$ calls $M_0$
- Machine $M'$ replaces Rule $R_{1j}$ with $n$ Rules:
  - $G_{ij}$ becomes: $G_{ij}$ and $G_{01}$, $G_{ij}$ and $G_{02}$, …, $G_{ij}$ and $G_{on}$
  - $E_{1j}$ becomes: $E_{1j}$; $E_{01}$; $E_{1j}$; $E_{02}$, …, $E_{1j}$; $E_{on}$

Semantics of Hierarchical Composition

Update Sets from Sub Machines are composed hierarchically, in a bottom up fashion, into a single Update Set before being applied to the Environment

Composition Operators $\otimes$ and $\oplus$:
- $\otimes$: Used to compose update sets from Sub machines at the same “level”
- $\oplus$: Used to compose update sets from Sub machines at different “levels”

Intuitively:
- All update sets done at same level are done in parallel, meaning duration will be max of durations and resource usages will be summations
- Time and resource annotations at higher levels override annotations at lower levels
Semantics of Hierarchical Composition - Example

\[ \text{US}_{\text{rel}} = \text{US}_1 \oplus (\text{US}_2 \oplus (\text{US}_5 \otimes \text{US}_6)) \otimes (\text{US}_3 \oplus \text{US}_7) \otimes \text{US}_4) \]

The TASM Language – Static Analysis II

- Theorem: Completeness is preserved through hierarchical composition
  - Proof Idea: Demorgan’s Law

- Theorem: Consistency is preserved through hierarchical composition
  - Proof Idea: Demorgan’s Law
The TASM Language – Parallel Composition

- A Main Machine is a unit of concurrency

- Time synchronizes Machines through step durations
  - All rules have same step durations > 0: synchronous multi-agent ASMs
  - All rules have same step durations = 0: asynchronous multi-agent ASMs

- Resource Consumption is additive through overlapping Update Sets

- A Machine is “busy” until the duration of its Rule execution elapses
The TASM Language – Parallel Composition Example

- Demo 2

The TASM Language – Other Constructs

- **else**
  - Special Rule enabled when no other Rule is enabled
- **now**
  - Used to get value of current time
- **t := next**
  - Used to wait until another machine makes a move
- **chan_name?**
  - Used for a machine to wait for a notification on a channel named “chan_name”
- **chan_name!**
  - Used to for a machine to send a notification along a channel named “chan_name”
The Hi-Five Framework + The TASM Language

- How to incorporate the state of the art
  - V & V
  - Test case generation

- How to reuse existing tools
  - Model checkers
  - SAT solvers

Static Analysis

- Completeness
  - Does the specified system have an answer for every possible class of inputs?

- Consistency
  - Does the specified system contain any “unwanted” non-determinism?

- Best-case and Worst-case behavior
  - Time
  - Resources
Analysis of TASM Specifications

- Translation to SAT
  - Completeness
  - Consistency

- Translation to UPPAAL
  - Worst-case execution time
  - Worst-case resource consumption
  - Best-case execution time
  - Best-case resource consumption

Translation to SAT - I

- For a given machine:
  - Completeness:
    - Is “not($G_1$ or $G_2$ or … or $G_n$)” satisfiable?
      - Yes: Incomplete -> Counterexample
      - No: Complete
  
  - Consistency:
    - Are all pairs “($G_i$ and $G_j$)” where $i, j >= 0, i != j$ satisfiable?
      - Yes: Inconsistent -> Counterexample
      - No: Consistent
Translation to SAT - II

- Translating Guards to Boolean Formulas:
  - Reduce integer constraints to a UDT using intervals
  - Create “at least one” clause
  - Create “at most one” clause
  - Translate resulting formula to DIMACS format

Translation to UPPAAL - I

- Map each Main Machine $M$ to a Timed Automaton

```
R_i
{t := [a, b];
  if Gi then
    E_i;
    CS_i;
    CR_i;
}
```
Translation to UPPAAL - II

- Else Rule uses urgent edge

```
R_i
{
    else then
    skip;
}
```

Translation to UPPAAL - III

- Verifying Execution Time: Iterative Bounded Liveness
  - Augment Model using boolean variable \( b \) and extra clock \( z \)
  - \( \Phi_{\text{init}} = E<> (b == 1) \)
  - \( \Phi_{\text{min}} = A[]((b == 1) \implies (z >= t)) \)
  - \( \Phi_{\text{max}} = A[]((b == 1) \implies (z <= t)) \)
  - Use \( \Phi_{\text{init}} \) to get initial value of \( t \)
  - Iterate \( \Phi_{\text{min}} \) until it is satisfied to get BCET
  - Iterate \( \Phi_{\text{max}} \) until it is satisfied to get WCET
The TASM Toolset I

- Toolset to:
  - Create and edit TASM specifications
  - Simulate TASM specifications
  - Analyze TASM specifications
  - Generate test cases based on TASM specifications

- Properties:
  - Open source (eventually)
  - Java based
  - Graphical Interface
  - Step-by-Step Simulation
  - XML API to export/import Specifications

The TASM Toolset II

- Integrates:
  - SAT4J Sat Solver
  - UPPAAL server

- Available from:
  - http://esl.mit.edu/tasm

- Now on Linux!
Case Studies

- Electronic Throttle Controller
- Production Cell
- Timeliner System
- N-Redundant Modular Avionics

The Production Cell (Revisited)
The Production Cell (Revisited)

<table>
<thead>
<tr>
<th>Component</th>
<th>Action</th>
<th>Time</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feed</td>
<td>Move block from beginning to end</td>
<td>5</td>
<td>500</td>
</tr>
<tr>
<td>Deposit</td>
<td>Move block from beginning to end</td>
<td>7</td>
<td>500</td>
</tr>
<tr>
<td>Robot</td>
<td>Rotate 30°</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>Robot</td>
<td>Pick up a block</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>Robot</td>
<td>Drop a block</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>Press</td>
<td>Stamp a block</td>
<td>11</td>
<td>1500</td>
</tr>
</tbody>
</table>

The Production Cell – TASM Model

- Components
  - 6 main machines (components + controller)
  - 4 function machines (macros)
  - 8 sub machines (composition)

- Resources
  - Power consumption

- Synchronization channels
  - Controller to Robot communication
  - Feed, Deposit, and Press sensors
TASM Toolset demo with the Production Cell (Revisited)

- Demo 3

Analysis of the Production Cell

- Any deadlocks in the system?
  - Completeness

- Any non-determinism in the system?
  - Consistency

- What is the fastest time to process n blocks?
  - BCET analysis

- What is the most power consumed?
  - Worst-case resource analysis
Other Case Studies - I

- The Timeliner System
  - Scripting Environment used on the International Space Station (ISS)
  - Used to automate astronaut tasks
  - Shares processor usage with other processes using fixed timeslices
  - Each timeslice corresponds to a “pass” of the Timeliner System

Other Case Studies - II

- The Timeliner System
  - How long should the timeslice be?
  - In other words: What is the maximum execution time for one “pass”
  - How about the minimum execution time?

- In TASM, use translation to UPPAAL:
  - WCET to go from (beginning of pass) to (end of pass)
  - BCET to go from (beginning of pass) to (end of pass)
### Other Case Studies - III

**The Timeliner System Analysis Results**

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>$z$ ($\mu$s)</th>
<th>$\Phi$</th>
<th>$\Phi_{max}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>8960</td>
<td>9300</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>9300</td>
<td>10800</td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>13085</td>
<td>13425</td>
<td>F</td>
</tr>
<tr>
<td>12</td>
<td>18885</td>
<td>20450</td>
<td>F</td>
</tr>
<tr>
<td>15</td>
<td>22735</td>
<td>-</td>
<td>T</td>
</tr>
</tbody>
</table>

- **WCET = 22735 $\mu$s**
- **BCET = 3250 $\mu$s**

### Other Case Studies - IV

- **The Timeliner System Analysis Results**

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>$z$ ($\mu$s)</th>
<th>$\Phi$</th>
<th>$\Phi_{min}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>8960</td>
<td>8730</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>8730</td>
<td>3250</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>3250</td>
<td>-</td>
<td>F</td>
</tr>
</tbody>
</table>
Other Case Studies – V

- N-Redundant Modular Avionics
  - Maximum bus usage
  - End-to-end latency
  - AADL language to describe Architecture
  - TASM for component-level analysis
  - How to test it?

Test Case Generation based on TASM Specifications

- Translation to SAT
  - Rule coverage through guard satisfiability (Grieskamp, Gargantini)

- Translation to UPPAAL
  - Generate test cases to exercise best case and worst case behavior
  - State coverage
  - Edge coverage (similar to rule coverage)
Supporting Material

- ETC paper at CSDUML in September 2006
- WiP paper at RTSS Conference in December 2006
- Language paper at RTNS Conference in March 2007
- Translation to SAT and Completeness and Consistency Analysis to be presented at MBT (ETAPS) in April 2007 (Gurevich)
- WCET and BCET analysis submitted to ECRTS 2007
- …
- Language Reference and Toolset available from http://esl.mit.edu/tasm

Summary and Conclusion

- Toolset is coming along
- Language has been stabilized and documented
- Key analytical engines have been identified with integration strategies
  - UPPAAL
  - SAT4J

http://www.uppaal.com

http://www.sat4j.org
Summary and Conclusion

- Documentation, publications, and toolset available from http://esl.mit.edu/tasm
- Contact: tasm@mit.edu