Seminar 2: System Modeling

October 19, 2012

Joseph Sifakis
RiSD Laboratory, EPFL
System modeling

- is an essential engineering activity
- consists in building virtual system prototypes (mathematical model, software prototype) rather than physical prototypes
- provides a basis for a priori validation of the designed systems conferring numerous advantages such as enhanced
  - explicability e.g. error diagnostics
  - predictability of system’s behavior

Two approaches

- Ad-hoc modeling widely used in industry for performance evaluation, rapid prototyping
- Formal modeling rooted in a well-defined model construction method. Formal models should be
  - faithful, that is whatever property is satisfied for the model holds for the real system
  - generated automatically
For hardware, it is easy to get faithful logical finite state models represented as systems of boolean equations.
For software this may be much harder ....
For mixed Software / Hardware systems there are no faithful modeling techniques as we have a poor understanding of how software and the underlying platform interact.

Ideally, a system model should be obtained as the composition of models of its application software and its execution platform.
Model-based Design

- Basic Models
- Component-based Modeling
- Discussion
Develop a holistic and rigorous approach to system design based on the use of models leading from requirements to implementations. Models are used to

- Study the interaction between the designed system and its physical environment
- Study architectural and implementation choices, in particular the interaction between the application software and the execution platform
- Derive validated implementations

Model-based design
- Is currently applied to the development of complex control, signal processing and communication systems where the application software can be derived from mathematical models
- Its application to general purpose systems raises hard theoretical problems e.g. consistency of the design flow

Existing modeling frameworks
- Matlab/Simulink, Modelica (control systems)
- UML, SysML, AADL (General purpose SW and systems engineering)
- SystemC, TLM (System-oriented extension of HDL)
THREE AIRCRAFT, A SINGLE MODEL, AND 80% COMMON CODE.

THAT'S MODEL-BASED DESIGN.
Model-based Design – Simplified Flow

1. Requirements
2. Application SW
3. Execution & External Envt Models
4. Mapping
5. Transformation
6. System Model
7. Transformation
8. Implementation Model
9. Execution Platform
10. Code Generation

Functional Correctness
Extra-functional Correctness
**Thesis:**
*A Model of the reactive system can be obtained by “composing” the application SW with models of its execution and the external environment*
Model-based Design – A Simple Example

External Environment

DSP

Reactive C

Event handler

Deadline constraint:
\[ t_{\text{out}} - t_{\text{in}} < d \]

Throughput constraint:
no buffer overflow
Model-based Design – A Simple Example

- Reactive C
- Intermediate C Code
- SAXO-RT
- Intermediate C Code
- SAXO
- Machine Description
- Target Machine executable code
- IC2TIC
- Timed (instrumented) C Code
- Event Handler Timed Model
- Environment Timed Model
- Analysis
- Target Machine executable code
- Timing Diagnostics
- Exec. Times
- Instrumented C Code
- Model-based Design
- Basic Models
- Component-based Modeling
- Discussion
A transition system

- **Q** : set of states
- **A** : set of actions
- \( \subseteq Q \times A \times Q \) : transition relation

Write \( q - a \rightarrow q' \) for \( (q, a, q') \in \)

- \( q_0 q_1 \ldots q_i \ldots \) such that \( q_i - a_i \rightarrow q_{i+1} \) is an execution sequence
- A **run** is a maximal execution sequence
Given two transition systems $TS_1 = (Q_1, A, \rightarrow_1)$, $TS_2 = (Q_2, A, \rightarrow_2)$, a **bisimulation** $\cong \subseteq Q_1 \times Q_2$ is a symmetric relation such that

$q_1 \cong q_2$ if

\[
\begin{align*}
q_1-a \rightarrow_1 q_1' & \text{ implies } \exists q_2' \ q_2-a \rightarrow_2 q_2' \text{ and } q_1' \cong q_2' \\
q_2-a \rightarrow_2 q_2' & \text{ implies } \exists q_1' \ q_1-a \rightarrow_1 q_1' \text{ and } q_1' \cong q_2'
\end{align*}
\]

$TS_1 \cong TS_2$ if $\forall q_1 \in Q_1 \ \exists q_2 \in Q_2 \ q_1 \cong q_2$ and $\forall q_2 \in Q_2 \ \exists q_1 \in Q_1 \ q_1 \cong q_2$
**Basic Models – Operational Semantics**

**Syntax**

\[ P ::= \text{skip} | x := E \mid \text{if } C \text{ then } P \text{ else } P \mid \text{while } C \text{ do } P \mid P ; P \]

An operational semantics defines a transition relation \( <P, s> \rightarrow <P', s'> \) by a set of rules where \( P, P' \) are programs and \( s, s' \) are valuations of program variables.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transition</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;\text{skip}, s&gt; \rightarrow \text{skip}, s&gt;)</td>
<td>(&lt;x := E, s&gt; \rightarrow \text{skip}, s[E/x ]&gt;)</td>
<td>(C(s) = \text{true})</td>
</tr>
<tr>
<td>(&lt;\text{if } C \text{ then } P \text{ else } P', s&gt; \rightarrow \text{if } C \text{ then } P \text{ else } P', s&gt;)</td>
<td>(&lt;\text{if } C \text{ then } P \text{ else } P', s&gt; \rightarrow \text{if } C \text{ then } P \text{ else } P', s&gt;)</td>
<td>(C(s) = \text{false})</td>
</tr>
<tr>
<td>(&lt;\text{while } C \text{ do } P, s&gt; \rightarrow \text{while } C \text{ do } P, s&gt;)</td>
<td>(&lt;\text{while } C \text{ do } P, s&gt; \rightarrow \text{skip}, s&gt;)</td>
<td>(C(s) = \text{true})</td>
</tr>
<tr>
<td>(&lt;P, s&gt; \rightarrow \text{skip}, s'&gt;)</td>
<td>(&lt;P, s&gt; \rightarrow &lt;P', s'&gt;)</td>
<td>(C(s) = \text{false})</td>
</tr>
<tr>
<td>(&lt;P; P', s&gt; \rightarrow &lt;P', s'&gt;)</td>
<td>(&lt;P; P', s&gt; \rightarrow &lt;P; P', s'&gt;)</td>
<td></td>
</tr>
</tbody>
</table>
Computer Science and Control Theory developed in the past, each for a different purpose.

Now they should be combined to develop a unified discipline marrying computing and physicality: Hybrid Systems or Cyber-physical Systems.

Hybrid systems can be used for modeling:
- Control systems in avionics, automotive, smart grids, trains, ...
- resource management in SoC, mobile systems, networks
Control: cause systems to behave in the desired manner by using feedback. Underlying theory based on analytic models e.g. differential equations, finite difference equations.

Computer Science/Programming: focuses on computing functions
- Get some input value
- Perform a (possibly unbounded) sequence of operations on the input...
- Produce output

As a rule, no interaction between the « computer » and the external environment during the computation.
Basic Models – Hybrid Systems: Thermostat

**Diagram:***

- **CONTROLLER**
  - \( \Theta = m \)
  - \( \Theta = M \)

- **ROOM**
  - ON, OFF
  - \( m \) \( M \)
  - \( \Theta' = -k \Theta \)
  - \( m \leq \Theta \)

- **HEATER**
  - \( \Theta = m \)
  - \( \Theta = M \)
  - \( \Theta' = -k(h - \Theta) \)
  - \( \Theta \leq M \)

**Equations:**

- \( \Theta = m \)
- \( \Theta = M \)
- \( \Theta' = -k \Theta \)
- \( m \leq \Theta \)
- \( \Theta' = -k(h - \Theta) \)
- \( \Theta \leq M \)
Basic Models – Hybrid Systems: Vending Machine

Controller

insert push coin_back coke

reset test

Timer x

Timer x

x:=0 x=10

x:=0 x<10

x<20 x:=0

x'=1

x:=0

xo

x<10

x'=1

true

x':=1

x'=1

coin_back

coin_back

insert push coin_back coke
Basic Models – Hybrid Systems: Temperature Control

REACTOR

RODS 1

RODS 2

CONTROLLER

Requirement: \( m \leq \Theta \leq M \) and rods can be reused after \( T \)
An operational semantics can be defined for Hybrid Systems based on Timed Transition Systems.

**Timed Transition System**
- $Q$: set of states
- $A$: set of actions

**Property: time additivity**

- $q - t_1 \rightarrow q_2 - t_2 \rightarrow q_3$
- $q - t \rightarrow q'$
- untimed transition
- $t_1 + t_2$

Example transition:

- $q - a \rightarrow q'$

---

**Hybrid Systems**

**Operational Semantics**

- Hybrid Systems can be modeled using Timed Transition Systems.

**Example:**

- $Q$ - set of states
- $A$ - set of actions
- Transition rules:
  - Untimed transition: $q - a \rightarrow q'$
  - Time step: $q - t \rightarrow q'$

**Property:**

- Time additivity: $q_1 - t_1 \rightarrow q_2 - t_2 \rightarrow q_3$
A run is a maximal sequence
\[ q_0 \ q_1 \ \ldots \ q_i \ \ldots \] such that \[ q_i - t_i \rightarrow q_{i+1} \] or \[ q_i - a_i \rightarrow q_{i+1} \]

- \[ \text{time}[q_0, q_i] = \sum_{j<i} t_j \]
- \[ q_0 \ q_1 \ \ldots \ q_i \ \ldots \] is time-divergent if \[ \forall k \in \mathbb{N} \ \exists i \ \text{time}[q_0, q_i] > k \]

Restriction to time divergent runs!
Basic Models – Timed Transition Systems: Discretization

\[ a \text{ TIMEOUT}[2] b : \]
execute \( a \) within 2 time units otherwise execute \( b \)

time unit 1

time unit 0.5

dense time
Hybrid systems combine analytic models (differential equations) and procedural models (automata)

**Analytic models**
- Are widely used in physical systems engineering and HW engineering
- Their behavior can be described by systems of equations expressing the next state of the system only a short time into the future as a function of the current state and the current input:
  \[ Y(t+1) = f(Y(t),X(t)) \quad \text{and} \quad \frac{dY}{dt} = f(Y(t),X(t)) \]
- They can be represented as networks of components, each one characterized by its transfer function

\[ X(t) \rightarrow \oplus \rightarrow Y(t+1)=X(t)\oplus Y(t) \]

\[ Y(t) \rightarrow \text{Unit Delay} \rightarrow \]
Basic Models – Analytic vs. Procedural Models

Matlab/Simulink

Copyright 1990-2005 The MathWorks, Inc.
Basic Models – Analytic vs. Procedural Models

From Analytic Models to Transition Systems

- To determine the state for all future times requires an iterative procedure many times—each advancing time a small step.
- Given an initial state and an input flow it is possible to determine the corresponding run.
- For continuous systems, the run is a set of points determining a trajectory.

**Difficulties:** some functions that have a simple analytic formulation may have complex computational counterparts.

\[
X(t) \rightarrow \text{Unit Delay} \rightarrow Y(t) = X(t-1)
\]

Timed Automaton
Languages based on Hybrid Systems can be used to model resources

- Resources are time-dependent quantities adequately updated: when an action is executed it consumes/liberates an amount of resources.
- **Resource-Consistency**: Incremental and parallel modification of resources in a model should agree with laws governing physical resources. There is a significant difference between model and physical time:
  - **Physical time**
    - Is monotonically increasing.
    - Its progress cannot be blocked.
  - **Model time**
    - can block or can involve Zeno runs (as in timed automata)
    - Deadline miss = deadlock or time-lock.

- Additional difficulties arise in resource modeling for models of distributed systems in particular because time is a global variable on which depend resource dynamics.
System analysis techniques assume resource-robustness: small change of resource parameters entail commensurable change of performance. Unfortunately, systems are not robust, in general.

For example, one would expect that a system model would exhibit worst performance for worst-case execution times of its actions.

- Performance degradation can be observed for increasing speed of the execution platform – Timing Anomaly
- Non determinism is one of the identified causes of such counter-intuitive behavior

We lack theory for guaranteeing resource-robustness

- performance should change monotonically with resources
- analysis for worst-case and best-case values of resource parameters suffice to determine performance bounds.
Model-based Design
Basic Models
Component-based Modeling
Discussion
Component-based Modeling

- Components are indispensable for enhanced productivity and correctness.
- Component composition lies at the heart of the parallel computing challenge.
- There is no Common Component Model - **Heterogeneity**.

Execution Platform
Thread-based programming

Actor-based programming

Software Engineering

Systems Engineering
Thread-based programming

Actor-based programming

Software Engineering

Systems Engineering
Component-based Modeling – Coordination

Coordinator (Controller)
Enforcing Property $P(q_1, q_2, q_3)$

PROTOCOL
Mutual Exclusion:  \( P = (w=0) \lor (r=0) \land (w \leq 1) \)

\[
\begin{align*}
do \\
request_r: w=0 & \rightarrow r:=r+1; \ start_r || \\
release_r: true & \rightarrow r:=r-1; \ finish_r || \\
request_w: w=0 \land (r=0) & \rightarrow w:=w+1; \ start_w || \\
release_w: true & \rightarrow w:=w -1; \ finish_w \\
\end{align*}
\]
The product behavior

$P(#s,#r)$

$#s=#r$ (synchronization)

$0 \leq #s - #r \leq 2$

True (no interaction)

$0 \leq #s - #r$ (asynchronous interaction)

Non determinism
There exists a large variety of mechanisms used to express coordination between components e.g. semaphores, monitors, locks, function call, asynchronous message passing rendezvous, broadcast ....

Is it possible to express component coordination in terms of composition operators? We need a unified composition paradigm for describing and analyzing the coordination between components in terms of tangible, well-founded and organized concepts and characterized by

- **Orthogonality**: clear separation between behavior and coordination constraints
- **Minimality**: uses a minimal set of primitives
- **Expressiveness**: achievement of a given functionality with a minimum of mechanism and a maximum of clarity

None of the existing component composition frameworks satisfies these requirements

- Some are formal such as process algebras e.g. CCS, CSP, pi-Calculus
- Other are ad hoc such as most frameworks used in software engineering e.g. ADL, or systems engineering e.g. SystemC
Hybrid Systems:

- runs are sequences of steps involving synchronous time progress
Component-based Modeling – Composition

Synchronous systems e.g. hardware, multimedia application SW,
- are an abstraction of timed models
- runs are sequences of steps involving strong synchronization
Component-based Modeling – Composition

Asynchronous systems e.g. general purpose SW, distributed systems

- No predefined execution step

Open problem: Theory for consistently composing hybrid, synchronous and asynchronous systems e.g. GALS
- Model-based Design
- Basic Models
- Component-based Modeling
- Discussion
In modern system design, modeling interaction of the application software with its execution and external environment is of paramount importance.

We need languages, models and tools supporting model-based design approaches for Cyber-physical systems.

Cyber-physical systems is an active research area identified as a priority by:

- The report of the US President's Council of Advisors on Science and Technology, December 2010
  [http://www.whitehouse.gov/sites/default/files/microsites/ostp/pcast-nitrd-report-2010.pdf](http://www.whitehouse.gov/sites/default/files/microsites/ostp/pcast-nitrd-report-2010.pdf)

- ACATECH, The German National Science and Engineering Academy

More on the BIP component framework can be found at
[http://www-verimag.imag.fr/Rigorous-Design-of-Component-Based](http://www-verimag.imag.fr/Rigorous-Design-of-Component-Based)
Thank You