The Adelaide Rosetta Project: Towards Simulation of Rosetta Descriptions

Peter Ashenden
Ashenden Designs and Adelaide University

Robert Esser and Joseph Kuehn
Adelaide University
What is Rosetta?

- Rosetta is a modeling language for representing heterogeneous information about a system.
- It supports different design domains using formal semantics appropriate for each domain:
  - models of computation and application-specific
  - semantics of interaction between domains
- Domains are used to describe facets of a component or system:
  - requirements, behaviour, constraints
- Formal semantics enable different forms of analysis:
  - formal verification, simulation, synthesis, constraint management
Where Did Rosetta Come From?

- SLDL Committee
  - established by EDA Industry Council, October 1996
  - complete requirements at www.inmet.com/SLDL/
- Language design contract let by US Air Force in 1999
  - Perry Alexander, University of Kansas
  - Dave Barton, AverStar/Titan, VA
- Other development and prototyping contracts
  - UKans, AverStar and EDAptive in USA
  - Adelaide University and Ashenden Designs in South Australia
Where is Rosetta Headed?

- SLDL Committee morphed into System-Level Design and Semantics Committee under Accellera
  - Rosetta subcommittee
    - developing draft standard language definition
  - Semantics subcommittee
    - developing semantic definitions for model-of-computation domains
- Rosetta draft standard will be put forward to IEEE or other accredited standards organization
  - around end-2002
Anatomy of a Rosetta Specification

— Perry Alexander, 2001
Facets and Domains

- A *facet* defines one view of a component
  - includes a *domain* that defines the universe of discourse for the facet: the semantic vocabulary
- Rosetta core domains define models of computation
- Simulation means different things in different domains
A Rossetta Example

```plaintext
facet autopilot ( vt, ht :: in real;  v, h :: in real;
P, d :: out real ) is
  // declarations
begin discrete_time
  // terms
end autopilot;
```

- A facet defines a view of a system or subsystem
- Parameters form the interface of the facet
A Rosetta Example

```vml
facet autopilot ( vt, ht :: in real; v, h :: in real;
P, d :: out real ) is

Pmax :: real is 100.0;
dmax :: real is pi/4;
Kw :: real is 0.05;
Kcv :: real is 1;
Kcd :: real is 0.5;
herr, hnorm, dunlim :: real;

limit ( v :: number; lower, upper :: number ) :: number is
  (v max lower) min upper;

begin discrete_time
  // terms
end autopilot;
```

- The declarations introduce local types, variables, functions, etc.
A Rosetta Example

```plaintext
facet autopilot ( vt, ht :: in real; v, h :: in real;
P, d :: out real ) is

Pmax :: real is 100.0;
dmax :: real is pi/4;
Kw :: real is 0.05;
Kcv :: real is 1;
Kcd :: real is 0.5;
herr, hnorm, dunlim :: real;

limit ( v :: number; lower, upper :: number ) :: number is
  (v max lower) min upper;

begin discrete_time
  // terms
end autopilot;
```

The domain included in the facet
A Rosetta Example

```plaintext
facet autopilot ( vt, ht :: in real; v, h :: in real;
    P, d :: out real ) is

    // declarations
    begin discrete_time
        P1: P@t0 = 0;
        P2: P' = limit(P + Kcv * (vt - v), -Pmax, Pmax);
        H1: d@t0 = 0;
        H2: herr = ht - h;
        H3: hnorm = if (herr < -pi) then (herr + 2*pi)
            else if (herr > pi) then (herr - 2*pi)
            else herr
            endif endif;
        H4: dunlim = -Kcd * hnorm;
        H5: d' = limit(dunlim, -dmax, dmax);

    end autopilot;
```

- The terms specify properties of the facet
- Terms are logical assertions
A Rosetta Example

facet autopilot ( vt, ht :: in real; v, h :: in real;
P, d :: out real ) is

// declarations
begin discrete_time
P1: P@t0 = 0;
P2: P' = limit(P + Kcv * (vt - v), -Pmax, Pmax);
H1: d@t0 = 0;
H2: herr = ht - h;
H3: hnorm = if (herr < -pi) then (herr + 2*pi)
else if (herr > pi) then (herr - 2*pi)
else herr
endif endif;
H4: dunlim = -Kcd * hnorm;
H5: d' = limit(dunlim, -dmax, dmax);
end autopilot;

- The discrete_time domain defines the @ and ’ operators
- x’ means the value of x in the next time step
- x means value of x in the current time step
**Discrete Domains**

- **Discrete_time** models time starting from $t_0$ and advancing in discrete steps
  - domain variable $t$ is the current time
  - $x_{@t+n}$ means value of $x$ after $n$ time steps from current time
  - $x'$ means $x_{@t+1}$

- **State_based** models systems that start in an initial state $s_0$ and advance through discrete states
  - domain variable $s$ is the current state
  - $x_{@s+n}$ means value of $x$ after $n$ states from current
  - $x'$ means $x_{@s+1}$
Simulating Discrete Domain Models

- Many discrete-domain models have terms of the form
  \[ x' = f(x, y, z, \ldots) \]
- Given initial values of variables at \( t_0 \) or \( s_0 \)
  - advance \( t \) or \( s \) step by step
  - for each step, calculate new values from values in previous step
- Forward driven
  - continue until stopping condition reached
- Lazy evaluation
  - request values of variables at some time/state
  - compute as needed to determine requested values
Data Flow Analysis

- For models that aren’t of this simple form
  - data flow analysis may reveal a dependency graph that allows computation to be done as t or s advance

- Example: in autopilot model
Rosetta Tool Architecture

- Front-end parser generates a semantic object model
- Back-end tools support various design capabilities
- At Adelaide, we are developing a native simulator for discrete-time and state-based models

---

— Perry Alexander, 2001
Autopilot Simulation

Evaluator: [autopilot]

<table>
<thead>
<tr>
<th>t</th>
<th>vt</th>
<th>v</th>
<th>P</th>
<th>ht</th>
<th>h</th>
<th>dunlim</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0+0</td>
<td>4.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.507</td>
<td>0.0</td>
<td>-0.785</td>
<td>0.0</td>
</tr>
<tr>
<td>t0+1</td>
<td>4.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.507</td>
<td>0.0</td>
<td>-0.785</td>
<td>-0.785</td>
</tr>
<tr>
<td>t0+2</td>
<td>4.0</td>
<td>0.4</td>
<td>0.0</td>
<td>1.507</td>
<td>0.0</td>
<td>-0.785</td>
<td>-0.785</td>
</tr>
<tr>
<td>t0+3</td>
<td>4.0</td>
<td>1.184</td>
<td>11.6</td>
<td>1.507</td>
<td>0.0157</td>
<td>-0.777</td>
<td>-0.785</td>
</tr>
<tr>
<td>t0+4</td>
<td>4.0</td>
<td>2.2038</td>
<td>14.416</td>
<td>1.507</td>
<td>0.0622</td>
<td>-0.754</td>
<td>-0.777</td>
</tr>
<tr>
<td>t0+5</td>
<td>4.0</td>
<td>3.1597</td>
<td>16.212</td>
<td>1.507</td>
<td>0.1478</td>
<td>-0.711</td>
<td>-0.754</td>
</tr>
<tr>
<td>t0+6</td>
<td>4.0</td>
<td>3.7824</td>
<td>17.052</td>
<td>1.507</td>
<td>0.2670</td>
<td>-0.651</td>
<td>-0.711</td>
</tr>
<tr>
<td>t0+7</td>
<td>4.0</td>
<td>4.0572</td>
<td>17.269</td>
<td>1.507</td>
<td>0.4016</td>
<td>-0.584</td>
<td>-0.651</td>
</tr>
<tr>
<td>t0+8</td>
<td>4.0</td>
<td>4.1380</td>
<td>17.212</td>
<td>1.507</td>
<td>0.5338</td>
<td>-0.518</td>
<td>-0.584</td>
</tr>
<tr>
<td>t0+9</td>
<td>4.0</td>
<td>4.1469</td>
<td>17.074</td>
<td>1.507</td>
<td>0.6547</td>
<td>-0.458</td>
<td>-0.513</td>
</tr>
<tr>
<td>t0+10</td>
<td>4.0</td>
<td>4.1347</td>
<td>15.927</td>
<td>1.507</td>
<td>0.7623</td>
<td>-0.404</td>
<td>-0.458</td>
</tr>
<tr>
<td>t0+11</td>
<td>4.0</td>
<td>4.1179</td>
<td>15.793</td>
<td>1.507</td>
<td>0.8569</td>
<td>Mu</td>
<td>-0.404</td>
</tr>
</tbody>
</table>

Important messages will be displayed here.

Targetting for evaluation: [vt@t0+10]...
Current Status and Future Work

- Simulator performs demand-driven evaluation of models in simple monotonic form
- Dealing with hierarchical models
  - elaboration of facet instances
    - requires expanding semantic object model to substitute facet body for facet instance
    - rebind item references in instantiated context
    - deal with closures of first-class function objects
- Evaluation of more complex monotonic models
  - more complete dataflow analysis to identify induction variables
Future Work

- Evaluation of constraints
  - inequalities over model variables
  - check for satisfaction as simulation proceeds
- Interaction with simulators in other domains
  - solvers for continuous-time models
  - requires development of interaction semantics between domains
- Formal verification using state-space exploration