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

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 Rosetta Example

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

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

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
  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

The terms are logical assertions

The domain included in the facet
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;

Discrete Domains

- **Discrete_time** models time starting from t0 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
  s0 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
  \[ x' = f(x, y, z, ...) \]
- Given initial values of variables at t0 or s0
  - 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

```
      P  v  h
     /   \ / \  
    P' \  /   /  
      \ /   /  
        herr
         / \
        hnorm \
         /   \ 
        dunlim \
         /     \ 
        d'     
```
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

---

Autopilot Simulation

Evaluator: [autopilot]

```
Variables:
vt, v, h, R, d : real;
herr : herror, distlim : real;
begin discrete_tsteps

GVL: vt=2.4;
GVL: v = if h < 10 then 4
    elseif h != 10 then 2
    else 10;
GVL: herror = h - distlim;
GVL: herror = if herror < 0 then 0
    else
        if herror > 10
            then 10
            else if

Evaluate:

<table>
<thead>
<tr>
<th>t</th>
<th>x</th>
<th>y</th>
<th>P</th>
<th>h</th>
<th>herror</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Important messages will be displayed here.
Targeting for evaluation: (vt=2.4;)
```

---
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