A modeling language for reconfigurable distributed hybrid systems

Fabian Kratz

OAS Colloquium

October 27, 2005
Outline

• Introduction
  – Hybrid systems & formal modeling

• Charon

• R-Charon: Extend Charon with reconfiguration
  – Target domain & reconfiguration
  – Syntax & Semantics
  – Application example

• Contribution & future work
Hybrid system

- Hybrid: consisting of diverse elements
  - Continuous as well as discrete
  - Behavior
  - State

Example
- Bouncing ball
Hybrid system – Practical view

- Sometimes limited to
  - Physical device + embedded computer
  - Computer controls physical device

- Examples
  - Cars, aircrafts, robots, …

- Complex systems
  - Require formal modeling
Formal modeling of hybrid systems

- Several languages exist
- Hybrid automata
  - Continuous + discrete variables
  - Continuous behavior (passage of time): States
  - Discrete behavior: State-switch
- Thermostat example

\[ d(T) = -T + 10 \]
\[ T \leq 21 \]
\[ T := 22 \]

\[ d(T) = -T + 38 \]
\[ T \geq 27 \]
Formal modeling of distributed HS: Charon

- Developed at Penn

- Individual components described as *Agents*
  - Consist of a number of parallel modes

- Individual behaviors described as *Modes* with *Sub modes*
  - Hierarchical hybrid automaton

- Charon model
  - Set of agents
  - Communication between agents
Charon example – Vehicle formations
Charon example - Vehicle formations

Agent: Car 2

Agent: Car 1

Catch up
input real dist
output real a
alg{ 1 < a < 2 }
inv{ dist > 30 }
dist < 50

Follow
input real dist
output real a
alg{ a == 0 }
inv{ 20 <= dist <= 50 }
dist > 50

Brake
input real dist
output real a
alg{ a == -1 }
inv{ dist <= 20 }
inv{ v > 0 }
dist > 50

local real a
input real dist, x
output real v

init
a := 0

Car 1
v_1

Car 2
v_2

dist < 20

20 < dist < 50

dist < 20

dist < 20

dist < 20

dist < 20

dist < 20

dist < 20

\text{diff\{ d(v) == a \} }
Charon example – Vehicle formations

$\text{dist} \rightarrow 25, \ a \leftrightarrow 0, \ldots$

Follow

\begin{align*}
    & \text{alg}\{ a \equiv 0 \} \\
    & \text{inv}\{ 20 \leq \text{dist} \leq 50 \}
\end{align*}

Catch up

\begin{align*}
    & \text{alg}\{ 1 < a < 2 \} \\
    & \text{inv}\{ \text{dist} > 30 \}
\end{align*}

Follow

\begin{align*}
    & \text{alg}\{ a \equiv 0 \} \\
    & \text{inv}\{ 20 \leq \text{dist} \leq 50 \}
\end{align*}

\[ \text{dist} > 50 \quad \text{dist} < 50 \]
Charon - Limitation

- Charon limited to
  - Fixed architecture
  - Agents known a priori
  - Communication known a priori

- Reconfiguration in hybrid systems
  - Ongoing miniaturization
  - “New” developments

- Extend Charon to support reconfiguration → R-Charon
  - Formal (Syntax & Semantics)
  - Limit changes to Charon
Reconfiguration?

- No fixed definition

- Reconfiguration through change in
  - Position (mobility)
  - Structure (architecture)
  - State
  - Behavior
  - Software

- Choose application domain
R-Charon application domain

• Large-scale transportation systems
  – Limited environment
  – Interacting moving entities
  – Examples: Highway systems, air-traffic control

• Reconfigurable modular robots
  – Large number of small, simple modules
  – Each module contains an embedded processor
  – Automatically (dis)connect modules
Modular robot example: Polybot
Reconfiguration in application domain

- Large-scale transportation systems
  - Enter / leave an environment
  - Group together
  - Communication with other’s in reach

- Reconfigurable modular robots
  - Physical (dis)connection of modules
  - Modules communicate with other modules in reach
Reconfiguration primitives in R-Charon

- Creation of new agents
- Destruction of existing agents
- Creation of new links between agents
  - Link ↔ physical or communication connection
- Destruction of existing links between agents
R-Charon syntax – Create / destroy agents

- Agents created / destroyed in mode transitions

- Charon mode transition
  - Guard
  - Action (list of assignments)

- R-Charon mode transition
  - Guard
  - Action
  - Creation / destruction operator
R-Charon syntax – Create / destroy links

- Communication in Charon
  - Connect input and output variables

- Communication in R-Charon
  - Accessible variables
  - Reference variables ("pointers")
  - Create / destroy links through assignments to reference variable
Charon semantics

- Two layers
  - Modes
  - Agents

- For each mode
  - State = Valuation of mode variables
  - Discrete, continuous, and environment steps

- For each agent
  - State = Valuation of agent variables
  - Possible steps: A combination of the steps of the modes in the agent
Charon – Mode Semantics

- Environment step relation
  - Transition in sub mode
  - Mode transition, or
- Discrete step relation
  - active sub mode
  - Constraints of mode + Continuous step relation
  - Constraints of mode + Continuous step relation
- Step relations $R$
  - $(q_1, q_2) \in R$
  - State = valuation
- $\preceq$ sub modes

Charon – Mode Semantics
R-Charon – Mode semantics

- Dynamic links come for free

- Extend discrete steps
  - Add create /destroy information

- Augment step relation
  - List with created / destroyed agents
  - $((q_1, q_2), L) \in R$
  - $L = \langle a_{created1}, a_{destroyed1}, \ldots \rangle$

- Mode semantics
  - Discrete, continuous, and environment step relations

M_1 \xrightarrow{\text{destroy}(v_{2_{\text{ref}}})} M_2
create(v_{1_{\text{ref}}}, \text{type, init});
Charon – Agent semantics

- \( \geq 0 \) top-level modes
- State = valuation \( q \)
- Steps \( q_1 \rightarrow q_2 \)
- Continuous step
  - All modes take a continuous step
- Discrete step
  - One mode takes a discrete step
- Parallel composition of agents
  - Merge parallel agents into a single agent
R-Charon – Agent semantics

- Need higher level!
  - Capture dynamic set of agents

- Explicitly define
  - Discrete,
  - Continuous, and
  - Environment step relations
  based on step relations of top-level modes

- Introduce system level

  System
  Agents
  Modes
R-Charon – System semantics

- **State** = (q, A)
- **Discrete step**
  - Discrete step of 1 agent in A
- **Continuous step** - "passage of time"
  - Continuous step of all agents in A
- **System update step**
  - Agent add step
  - Agent delete step
- **Trace**:

\[
(q_1, A) \xrightarrow{C} (q_2, A) \xrightarrow{D,L} (q_3, A) \xrightarrow{U_a} (q_4, A \cup \{a_{new}\}) \xrightarrow{U_d} (q_4, A - \{a_{exist}\}) \xrightarrow{\cdots}
\]
R-Charon semantics – Difficulties

• Agents creation / destruction triggered at mode level
  – Operation in a mode transition
  – Impact at system level

• Created agents can create new agents

• Links to destroyed agents
R-Charon – Compositionality

- Proven for Charon
  - Based on trace semantics / refinement notion
  - Mode & agent

- Not proven for R-Charon

- Modes
  - More or less straightforward

- Agents
  - What does compositionality mean now?
  - System notion $\leftrightarrow$ parallel composition
R-Charon – Application example

- Air-traffic control

Nowadays: Follow waypoints

Future: Free flight
R-Charon – Application example

• Agents
R-Charon – Application example

Airspace

Wait

inv{ True }

create( newPlane, ..., Airplane )

System

Airspace

Airplane 1

Airplane 2

Airplane N

Airplane N + 1
R-Charon – Application example

- Collision avoidance

Airplane

- Normal operation
  - Straight to target

CAV guided
  - Fly around other plane

Airspace

- dist(pos, target) < 1
- destroy(this)
Conclusion – Research contribution

- R-Charon
  - Extends Charon with reconfiguration
  - Focused on two application domains
  - Formal syntax & semantics

- Charon models can still be used with minor notational changes

- Nice first step
  - Some insight into reconfiguration
Future work

• Case studies

• Location model

• Tooling
  – Extend Charon toolkit
  – Simulation of R-Charon models

• Reconfiguration within an agent
Questions