RPL: A Policy Language For Dynamic Reconfiguration

SERENE08

Richard Payne

Newcastle University

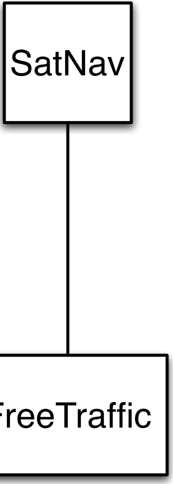
18 November 2008

Outline

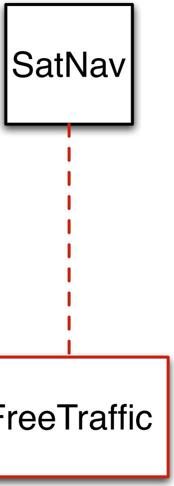
Background

- Open component-based systems
  - Components enter and leave system's environment.
  - Reacting to changes in environment - possibility of the system to be resilient
- System designer specifies requirements
  - Availability, mean-time-to-failure (MTTF), ...
- Alter configuration using *reconfiguration strategy*.
  - Switching components
  - Fault-tolerance mechanism

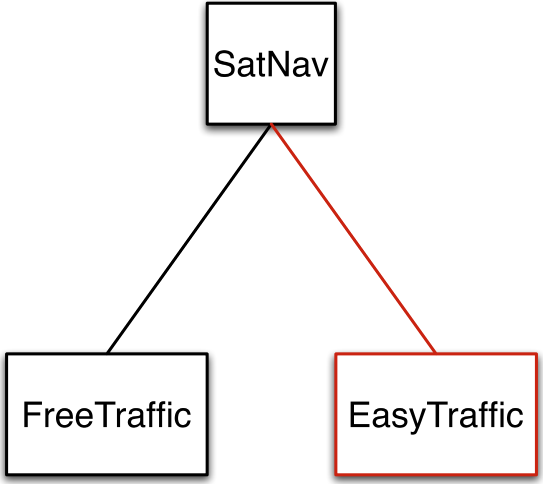
Reconfiguration Example



Reconfiguration Example



Reconfiguration Example



Problem

- Changes taking place need to be predictable at design time
- Reconfiguration policies needed
  - Represented in concise and clear structure
  - Formal language to reason over policies
  - Prove properties of policies

Properties

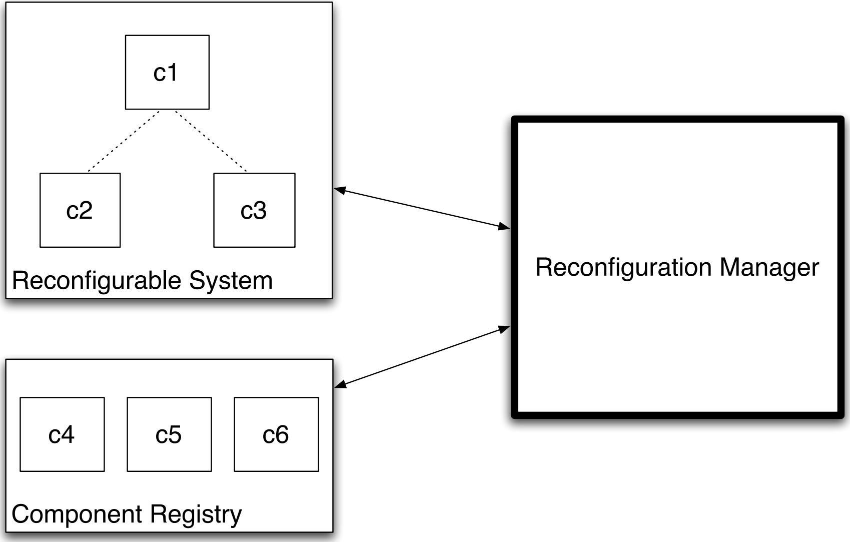
What sort of properties do we want these policies to respect?

- A policy should cover all non-optimal states a system can enter
- For a system in a given initial state, rules of a policy must be reachable - i.e. rules must be non-redundant
- Reconfiguration actions should not leave the system in an undesirable state
- If it is possible, the system should be able to perform at an optimal level of service

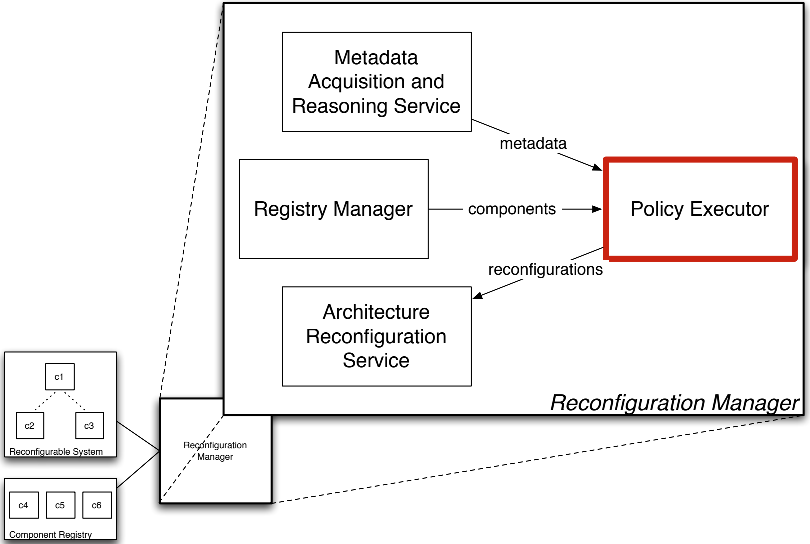
Approach

- Design a policy-based language
  - Require an architectural model
  - Require syntax and semantics
- Represent Properties
  - Decide what properties are important
  - Need to be formally defined
- Prove properties of policies
  - Do the policies created respect properties?

Framework


c1
c2
c3
Reconfigurable System
Reconfiguration Manager
c4
c5
c6
Component Registry

Framework


Metadata Acquisition and Reasoning Service
metadata
Registry Manager
components
Policy Executor
reconfigurations
Architecture Reconfiguration Service
Reconfiguration Manager
c1
c2
c3
Reconfigurable System
c4
c5
c6
Component Registry
Reconfiguration Manager

Policy Language v1.

- Policies represented as the whole map of conditions to responses
  - *if* condition *then* response
  - When system changes, need to ensure correct action is taken
- Difficulty occurs when ordering conditions
  - Non-deterministic
  - Priority bands
- How to represent resilience?

Teleo-Reactive Programs

- Created by Nils Nilsson
- Agent control program that directs an agent towards a goal, under a changing environment
- Based on autonomous agents such as mobile robots

Teleo-Reactive Programs

$$
\begin{aligned}
c_1 &\longrightarrow nil \\
c_2 &\longrightarrow a_2 \\
c_3 &\longrightarrow a_3 \\
&\vdots \\
c_{n-1} &\longrightarrow a_{n-1} \\
c_n &\longrightarrow a_n
\end{aligned}
$$

Teleo-Reactive Programs

Goal condition

$$
\begin{aligned}
c_1 &\rightarrow nil \\
c_2 &\rightarrow a_2 \\
c_3 &\rightarrow a_3 \\
&.. \\
&.. \\
c_{n-1} &\rightarrow a_{n-1} \\
c_n &\rightarrow a_n
\end{aligned}
$$

Teleo-Reactive Programs

$$
\begin{aligned}
c_1 &\rightarrow nil \\
c_2 &\rightarrow a_2 \\
c_3 &\rightarrow a_3 \\
&.. \\
&.. \\
c_{n-1} &\rightarrow a_{n-1} \\
c_n &\rightarrow a_n
\end{aligned}
$$

nil action

Teleo-Reactive Programs

Degraded conditions

$$
\begin{aligned}
c_1 &\rightarrow nil \\
c_2 &\rightarrow a_2 \\
c_3 &\rightarrow a_3 \\
&\vdots \\
c_{n-1} &\rightarrow a_{n-1} \\
c_n &\rightarrow a_n
\end{aligned}
$$

Teleo-Reactive Programs

$$
\begin{aligned}
c_1 &\longrightarrow nil \\
c_2 &\longrightarrow a_2 \\
c_3 &\longrightarrow a_3 \\
&\vdots \\
c_{n-1} &\longrightarrow a_{n-1} \\
c_n &\longrightarrow a_n
\end{aligned}
$$

Actions with aim to reach higher condition

Teleo-Reactive Example

SatNav Example

- $avail, cost$ - sensed metadata
- $MinAvail, MaxCost$ - designer specified bounds
- $reduceCost, increaseAvail$ - actions

Teleo-Reactive Example

SatNav Example

- $avail, cost$ - sensed metadata
- $MinAvail, MaxCost$ - designer specified bounds
- $reduceCost, increaseAvail$ - actions

$$avail \geq MinAvail \wedge cost \leq MaxCost \quad \rightarrow nil$$

Teleo-Reactive Example

SatNav Example

- $avail, cost$ - sensed metadata
- $MinAvail, MaxCost$ - designer specified bounds
- $reduceCost, increaseAvail$ - actions

$$avail \geq MinAvail \wedge cost \leq MaxCost \quad \rightarrow nil$$
$$avail \geq MinAvail \wedge cost > MaxCost \quad \rightarrow reduceCost$$

Teleo-Reactive Example

SatNav Example

- $avail, cost$ - sensed metadata
- $MinAvail, MaxCost$ - designer specified bounds
- $reduceCost, increaseAvail$ - actions

$$
\begin{array}{ll}
avail \geq MinAvail \wedge cost \leq MaxCost & \rightarrow nil \\
avail \geq MinAvail \wedge cost > MaxCost & \rightarrow reduceCost \\
avail < MinAvail & \rightarrow increaseAvail
\end{array}
$$

Teleo-Reactive Programs

- Suitable as a policy language
    - Meets definition of policy
    - Concise and structured ordering
- Suitable for resilience
    - Notion of degradation
    - Actions to return to optimal level of service
- No formal semantics

Structural Operational Semantics - Example

Syntax

$$
\begin{aligned}
\mathit{If} \;::\; & \mathit{test} : \mathit{Expr} \\
& \mathit{then} : \mathit{Stmt} \\
& \mathit{else} : \mathit{Stmt}
\end{aligned}
$$

Structural Operational Semantics - Example

Syntax

$$If \;::\; test : Expr \quad then : Stmt \quad else : Stmt$$

Semantic Definition

$$\text{If-T}\;\frac{(test, \sigma) \xrightarrow{e} \text{TRUE};\quad (then, \sigma) \xrightarrow{s} \sigma';}{(mk\text{-}If(test, then, else), \sigma) \xrightarrow{s} \sigma'}$$

Structural Operational Semantics - Example

Syntax

$$\begin{aligned} \mathit{If} \ :: \ & \mathit{test}: \mathit{Expr} \\ & \mathit{then}: \mathit{Stmt} \\ & \mathit{else}: \mathit{Stmt} \end{aligned}$$

Semantic Definition

$$\text{If-T}\ \frac{(\mathit{test}, \sigma) \xrightarrow{e} \textsc{True}; \quad (\mathit{then}, \sigma) \xrightarrow{s} \sigma';}{(\mathit{mk\text{-}If}(\mathit{test}, \mathit{then}, \mathit{else}), \sigma) \xrightarrow{s} \sigma'}$$

Structural Operational Semantics - Example

Syntax

$$\begin{aligned} \mathit{If} \;::\; & \mathit{test} : \mathit{Expr} \\ & \mathit{then} : \mathit{Stmt} \\ & \mathit{else} : \mathit{Stmt} \end{aligned}$$

Semantic Definition

$$\text{If-T}\ \frac{(\mathit{test}, \sigma) \xrightarrow{e} \textsc{True};\quad (\mathit{then}, \sigma) \xrightarrow{s} \sigma';}{(\mathit{mk\text{-}If}(\mathit{test}, \mathit{then}, \mathit{else}), \sigma) \xrightarrow{s} \sigma'}$$

Structural Operational Semantics - Example

Syntax

$$
\begin{aligned}
If \quad :: \quad & test : Expr \\
& then : Stmt \\
& else : Stmt
\end{aligned}
$$

Semantic Definition

$$
\text{If-T} \; \frac{(test, \sigma) \stackrel{e}{\longrightarrow} \textsc{True}; \quad (then, \sigma) \stackrel{s}{\longrightarrow} \sigma';}{(mk\text{-}If(test, then, else), \sigma) \stackrel{s}{\longrightarrow} \sigma'}
$$

Structural Operational Semantics - Example

Syntax

$$\begin{aligned} If \ :: \ & test : Expr \\ & then : Stmt \\ & else : Stmt \end{aligned}$$

Semantic Definition

$$\text{If-T} \ \frac{(test, \sigma) \xrightarrow{e} \textsc{True}; \quad (then, \sigma) \xrightarrow{s} \sigma';}{(mk\text{-}If(test, then, else), \sigma) \xrightarrow{s} \sigma'}$$

Teleo-Reactive Syntax

$$
\begin{aligned}
TRSpecification \;::\;& vars\colon Id \xrightarrow{m} ScalarType \\
& trseq\colon TRSequence \\
& acts\colon Id \xrightarrow{m} Action
\end{aligned}
$$

$$TRSequence \;::\; rules\colon TRRule^*$$

$$
\begin{aligned}
TRRule \;::\;& cond\colon Expression \\
& resp\colon Response
\end{aligned}
$$

$$Response = Id \mid TRSequence$$

Teleo-Reactive Semantics - State

State representation

$$\Sigma = Id \xrightarrow{m} ScalarInfo$$

Teleo-Reactive Semantics - State

State representation

$$\Sigma = Id \xrightarrow{m} ScalarInfo$$

$$ScalarInfo \;::\; scope\colon \textsc{World} \mid \textsc{Internal}$$
$$value\colon ScalarValue$$

Teleo-Reactive Semantics - TRSequence

$$\stackrel{trs}{\rightarrow}: (\,TRSequence \times \Sigma) \times Id.$$

Teleo-Reactive Semantics - TRSequence

$$\xrightarrow{trs} : (TRSequence \times \Sigma) \times Id.$$

$$\text{TRSeq}\quad \frac{\begin{array}{l} trseq' = [trseq(i) \mid i \in \textbf{inds}\ trseq \cdot \\ \qquad (trseq(i).cond, \sigma) \xrightarrow{e} \text{TRUE}]; \\ trseq' \neq [\,]; \\ rule = \textbf{hd}\ trseq'; \\ mk\text{-}TRRule(cond, resp) = rule; \\ (resp, \sigma) \xrightarrow{r} a \end{array}}{(trseq, \sigma) \xrightarrow{trs} a}$$

Teleo-Reactive Semantics - TRSequence

$$\stackrel{trs}{\longrightarrow} : (TRSequence \times \Sigma) \times Id.$$

$$\begin{array}{l}
trseq' = [trseq(i) \mid i \in \textbf{inds}\ trseq \cdot \\
\qquad (trseq(i).cond, \sigma) \stackrel{e}{\longrightarrow} \textsc{True}]; \\
trseq' \neq [\,]; \\
rule = \textbf{hd}\ trseq'; \\
mk\text{-}TRRule(cond, resp) = rule; \\
(resp, \sigma) \stackrel{r}{\longrightarrow} a \\
\hline
(trseq, \sigma) \stackrel{trs}{\longrightarrow} a
\end{array} \quad \boxed{\text{TRSeq}}$$

Teleo-Reactive Semantics - TRSequence

$$\xrightarrow{trs}: (TRSequence \times \Sigma) \times Id.$$

$$\text{TRSeq}\quad \frac{\begin{array}{l} trseq' = [trseq(i) \mid i \in \textbf{inds}\ trseq \cdot \\ \qquad (trseq(i).cond, \sigma) \xrightarrow{e} \text{TRUE}]; \\ trseq' \neq [\,]; \\ rule = \textbf{hd}\ trseq'; \\ mk\text{-}TRRule(cond, resp) = rule; \\ (resp, \sigma) \xrightarrow{r} a \end{array}}{(trseq, \sigma) \xrightarrow{trs} a}$$

Teleo-Reactive Semantics - TRSequence

$$\xrightarrow{trs}: (\mathit{TRSequence} \times \Sigma) \times \mathit{Id}.$$

$$\frac{\begin{array}{l} trseq' = [trseq(i) \mid i \in \textbf{inds}\ trseq \cdot \\ \qquad (trseq(i).cond, \sigma) \xrightarrow{e} \mathsf{TRUE}]; \\ trseq' \neq [\,]; \\ rule = \textbf{hd}\ trseq'; \\ mk\text{-}TRRule(cond, resp) = rule; \\ (resp, \sigma) \xrightarrow{r} a \end{array}}{(trseq, \sigma) \xrightarrow{trs} a}\ \boxed{\text{TRSeq}}$$

Teleo-Reactive Semantics - TRSequence

$$\xrightarrow{trs}: (TRSequence \times \Sigma) \times Id.$$

$$\text{TRSeq}\quad \frac{\begin{array}{l} trseq' = [trseq(i) \mid i \in \textbf{inds}\ trseq \cdot \\ \qquad (trseq(i).cond, \sigma) \xrightarrow{e} \textsf{TRUE}]; \\ trseq' \neq [\,]; \\ rule = \textbf{hd}\ trseq'; \\ mk\text{-}TRRule(cond, resp) = rule; \\ (resp, \sigma) \xrightarrow{r} a \end{array}}{(trseq, \sigma) \xrightarrow{trs} a}$$

Teleo-Reactive Semantics - TRSequence

$$\xrightarrow{trs} : (TRSequence \times \Sigma) \times Id.$$

$$\frac{\begin{array}{l} trseq' = [trseq(i) \mid i \in \mathbf{inds}\ trseq \cdot \\ \qquad (trseq(i).cond, \sigma) \xrightarrow{e} \text{TRUE}]; \\ trseq' \neq [\,]; \\ rule = \mathbf{hd}\ trseq'; \\ mk\text{-}TRRule(cond, resp) = rule; \\ (resp, \sigma) \xrightarrow{r} a \end{array}}{(trseq, \sigma) \xrightarrow{trs} a} \ \boxed{\text{TRSeq}}$$

Discussion

- Issues with semantic definition
    - Handling of interrupts
    - No notion of time
    - How feasible are proofs?

Future Work

- Complete case study
- Compare semantic approaches to T-R language definition
  - Hayes' Time-Interval approach
- Extend T-R language for reconfiguration actions
  - Architecture representation
  - Reconfiguration actions
- Prove properties of policies

The End.

Thank you

Teleo-Reactive Semantics - TRSpecification

$$\sigma = \{id \mapsto mk\text{-}ScalarInfo(vars(i).scope, 0) \mid id \in \mathbf{dom}\ vars \cdot vars(id).type = \text{NAT}\} \vee \{id \mapsto mk\text{-}ScalarInfo(vars(i).scope, \text{TRUE}) \mid id \in \mathbf{dom}\ vars \cdot vars(id).type = \text{BOOL}\};$$

$$trprog = mk\text{-}TRProg(trseq, acts);$$

$$(trseq, \sigma) \xrightarrow{trs} a;$$

$$(a, trprog, \sigma) \xrightarrow{c} \sigma'$$

TRSpec

$$\overline{(mk\text{-}TRSpecification(vars, trseq, acts)) \xrightarrow{trsp} \sigma'}$$

Teleo-Reactive Semantics - Response

$$\boxed{\text{ActResponse}}\ \frac{a \in Id}{(a, \sigma) \xrightarrow{r} a}$$

$$\boxed{\text{TRResponse}}\ \frac{(trseq, \sigma) \xrightarrow{trs} a}{(trseq, \sigma) \xrightarrow{r} a}$$

Teleo-Reactive Semantics - ActionControl

$$\text{ActCont1}\quad \frac{\begin{array}{l} act = trprog.acts(a); \\ (act, \sigma) \xrightarrow{a} \sigma'; \\ \forall i \in \mathbf{dom}\, \sigma \cdot \sigma(i).scope = \textsc{World} \\ \qquad\qquad \wedge\ \sigma(i).value = \sigma'(i).value; \\ (a, trprog, \sigma') \xrightarrow{c} \sigma'' \end{array}}{(a, trprog, \sigma) \xrightarrow{c} \sigma''}$$

Teleo-Reactive Semantics - ActionControl

$$
\textsf{ActCont1}\;
\frac{\begin{array}{l}
act = trprog.acts(a);\\
(act, \sigma) \xrightarrow{a} \sigma';\\
\forall i \in \textbf{dom}\, \sigma \cdot \sigma(i).scope = \textsc{World}\\
\qquad\qquad \wedge\, \sigma(i).value = \sigma'(i).value;\\
(a, trprog, \sigma') \xrightarrow{c} \sigma''
\end{array}}
{(a, trprog, \sigma) \xrightarrow{c} \sigma''}
$$

Teleo-Reactive Semantics - ActionControl

$$\frac{\begin{array}{l} act = trprog.acts(a); \\ (act, \sigma) \xrightarrow{a} \sigma'; \\ \forall i \in \mathbf{dom}\, \sigma \cdot \sigma(i).scope = \textsc{World} \\ \qquad\qquad \wedge\, \sigma(i).value = \sigma'(i).value; \\ {\color{red}(a, trprog, \sigma') \xrightarrow{c} \sigma''} \end{array}}{(a, trprog, \sigma) \xrightarrow{c} \sigma''} \; \boxed{\text{ActCont1}}$$

Teleo-Reactive Semantics - ActionControl

$$\frac{\begin{array}{l} act = trprog.acts(a); \\ (act, \sigma) \xrightarrow{a} \sigma'; \\ \exists i \in \textbf{dom}\, \sigma \cdot \sigma(i).scope = \textsc{World} \\ \qquad \wedge\, \sigma(i).value \neq \sigma'(i).value; \\ (trprog.seq, \sigma') \xrightarrow{trs} a'; \\ (a', trprog, \sigma') \xrightarrow{c} \sigma'' \end{array}}{(a, trprog, \sigma) \xrightarrow{c} \sigma''} \; \text{ActCont2}$$

Teleo-Reactive Semantics - ActionControl

$$\text{ActCont2}\quad \frac{\begin{array}{l} act = trprog.acts(a);\\ (act, \sigma) \xrightarrow{a} \sigma';\\ \exists i \in \mathbf{dom}\, \sigma \cdot \sigma(i).scope = \textsc{World}\\ \qquad\qquad \wedge\, \sigma(i).value \neq \sigma'(i).value;\\ (trprog.seq, \sigma') \xrightarrow{trs} a';\\ (a', trprog, \sigma') \xrightarrow{c} \sigma'' \end{array}}{(a, trprog, \sigma) \xrightarrow{c} \sigma''}$$

Teleo-Reactive Semantics - ActionControl

$$
\text{ActCont2}\quad
\frac{
\begin{array}{l}
act = trprog.acts(a);\\
(act, \sigma) \xrightarrow{a} \sigma';\\
\exists i \in \mathbf{dom}\,\sigma \cdot \sigma(i).scope = \textsc{World}\\
\qquad\qquad \wedge\ \sigma(i).value \neq \sigma'(i).value;\\
(trprog.seq, \sigma') \xrightarrow{trs} a';\\
(a', trprog, \sigma') \xrightarrow{c} \sigma''
\end{array}
}{
(a, trprog, \sigma) \xrightarrow{c} \sigma''
}
$$

Teleo-Reactive Semantics - ActionControl

$$\frac{\begin{array}{l} act = trprog.acts(a); \\ (act, \sigma) \xrightarrow{a} \sigma'; \\ \exists i \in \mathbf{dom}\, \sigma \cdot \sigma(i).scope = \textsc{World} \\ \qquad\qquad \wedge\, \sigma(i).value \neq \sigma'(i).value; \\ (trprog.seq, \sigma') \xrightarrow{trs} a'; \\ (a', trprog, \sigma') \xrightarrow{c} \sigma'' \end{array}}{(a, trprog, \sigma) \xrightarrow{c} \sigma''} \;\text{ActCont2}$$