SEFM 2017

… the challenge of change …

Jeff Kramer

Imperial College London

… the challenge of change …

environment **E**
goals **G**
capabilities **I** of the system x

…. to be aware of and monitor these sources of change

off-line software evolution	⟷	***run-time software adaptation***
requirements analysis, design, implementation, redeployment ….		

… the challenge of change …

Off-line software **evolution**	⟷	Run-time software **adaptation**
pre-planned change (maintenance)		*unforeseen change*

.. the challenge is to
automate and run on-line
what is currently performed off-line!

Self-Managed Adaptive Systems

Adaptive *light* :

self adjustment of runtime parameters in response to degraded performance or failure

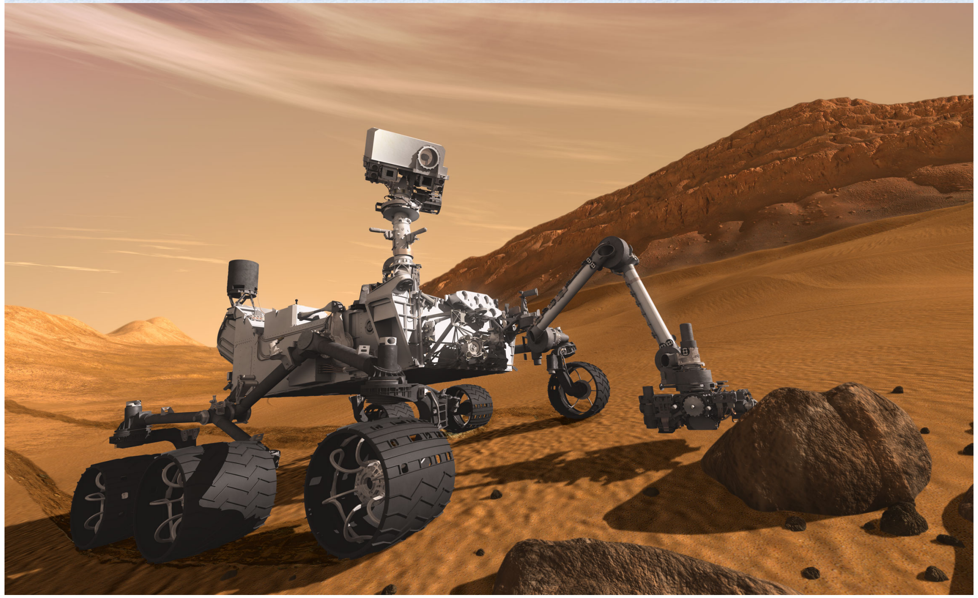
Adaptive *full fat* :

self change in functionality and performance in response to **unforeseen** changes in the environment, goals and/or capabilities of the system

Self-Managed

Disruptive change!

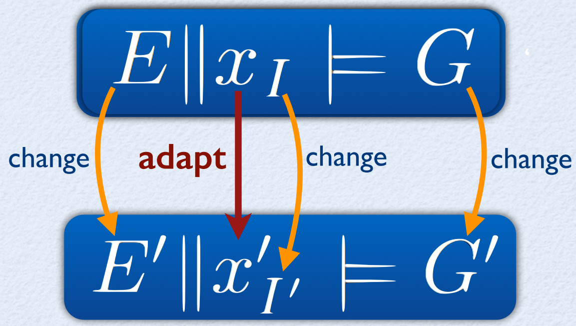
Self-Managed Adaptive Systems



Self-Managed Adaptive Systems

… more formally …

E - assumed environment behaviour
G - requirements goals of system
I - interface capabilities of the system **x**



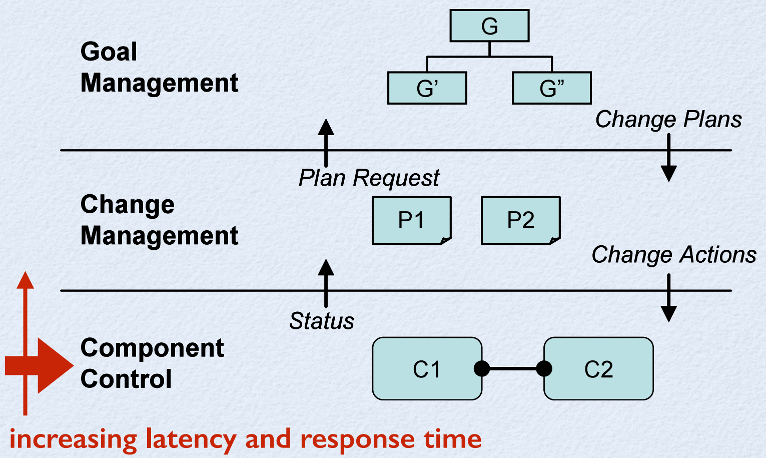
Self-Managed Adaptive Systems

models @ runtime

… in an appropriate
architecture
with a rich runtime
environment

architecture is important

three layer architecture



1. Planning over abstract domain

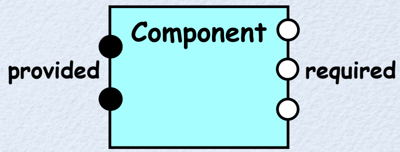
2. Precomputed plans: component assembly and plan execution

3. Component execution and dynamic configuration

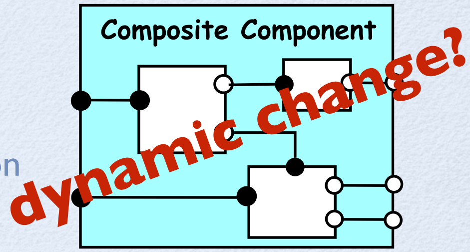
- a separation of concerns

CONIC and Darwin

- distributable, context-independent components
  - interaction via a well-defined interface



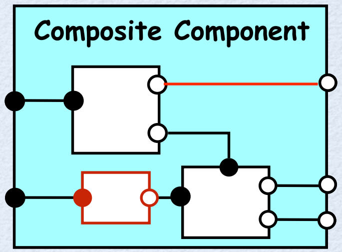
- an explicit configuration description (ADL)
  - third party instantiation and binding



TSE 1985, TSE 1989, ESEC/FSE 1995, FSE 1996

CONIC and Darwin

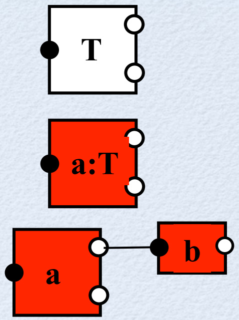
- on-line dynamic change
  - once installed, the software could be dynamically modified without stopping the entire system



TSE 1985, TSE 1989, ESEC/FSE 1995, FSE 1996

on-line dynamic change

- **load** component type
- **create/delete** component instances
- **bind/unbind** component services

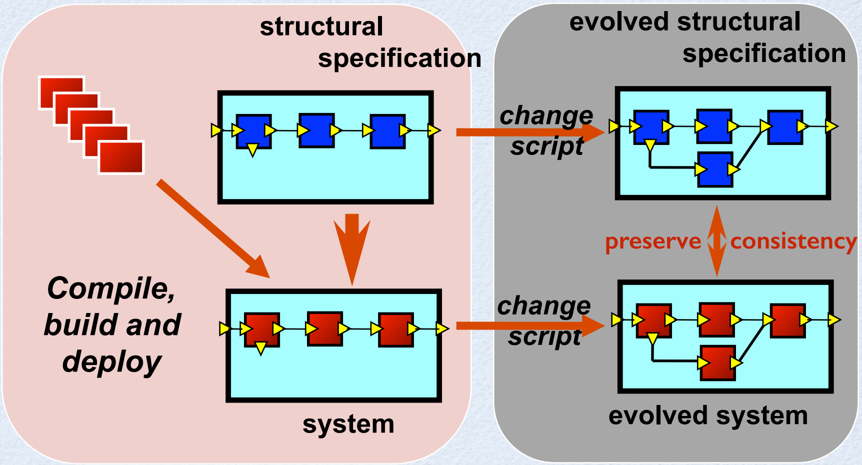


How can we do this ***safely?***

How can we maintain ***configuration consistency*** and ***behaviour consistency*** during the change?

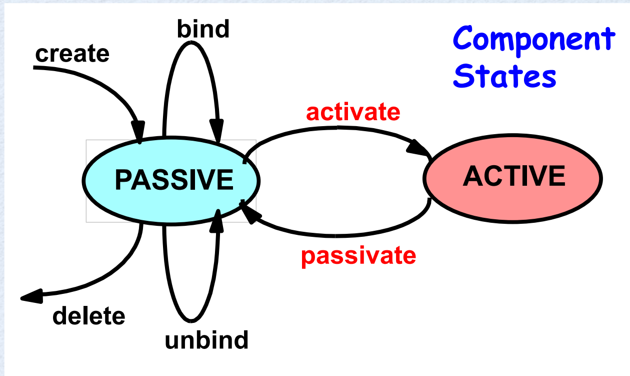
TSE 1985

configuration consistency



TSE 1985

behaviour consistency



General change model:

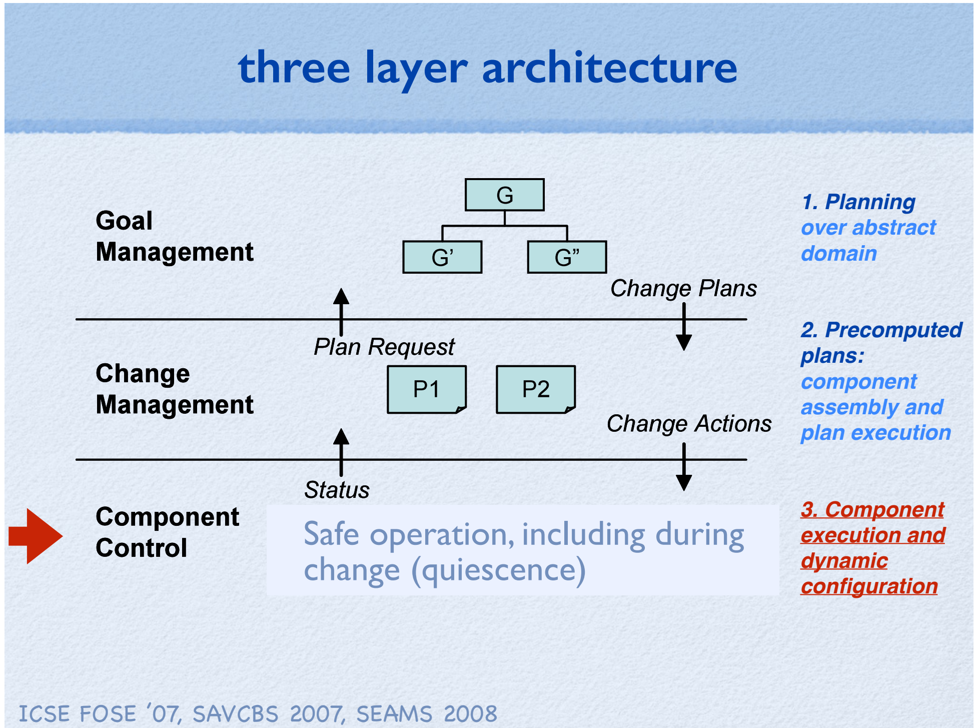
Separate the specification of *configuration change* from the component *application behaviour*.

Passive: the component services interactions, but does not initiate new ones i.e. acts to preserve consistency.

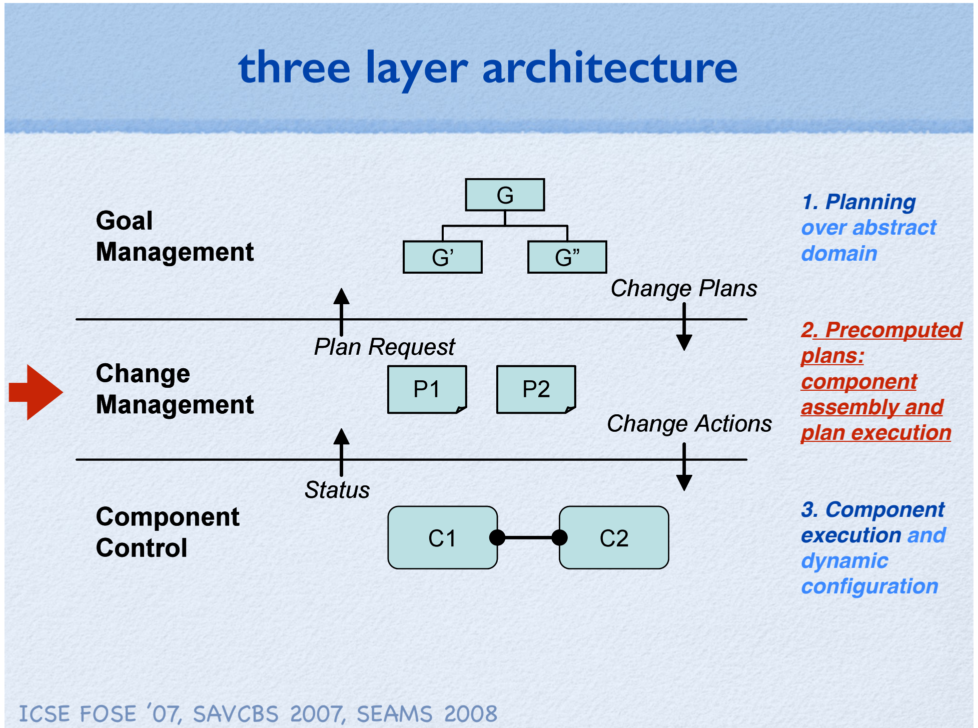
Quiescence : the component is passive and the environment is passive ie. no transactions will be initiated on it.

TSE 1990

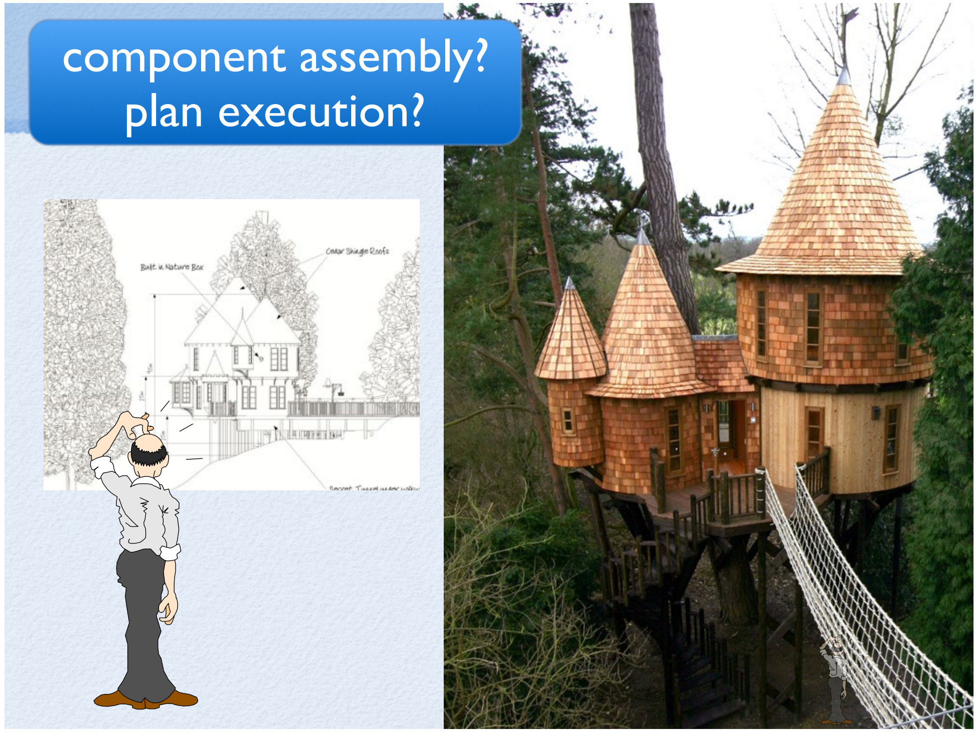
safe configuration and
reconfiguration of components
No components? use objects and dependency injection
(inversion of control) for 3rd party instantiation and binding!



three layer architecture
Goal Management
G
G'
G"
Change Plans
Plan Request
Change Management
P1
P2
Change Actions
Status
Component Control
Safe operation, including during change (quiescence)
1. Planning over abstract domain
2. Precomputed plans: component assembly and plan execution
3. Component execution and dynamic configuration
ICSE FOSE '07, SAVCBS 2007, SEAMS 2008



three layer architecture
Goal Management
G
G'
G"
Change Plans
Plan Request
Change Management
P1
P2
Change Actions
Status
Component Control
C1
C2
1. Planning over abstract domain
2. Precomputed plans: component assembly and plan execution
3. Component execution and dynamic configuration
ICSE FOSE '07, SAVCBS 2007, SEAMS 2008



component assembly?
plan execution?

plan execution

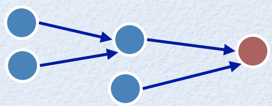
plan execution

```
...
AT.loc1 && !LOADED
       -> pickup
AT.loc1 && LOADED
       -> moveto.loc2
AT.loc2 && LOADED
       -> putdown
AT.loc2 && !LOADED
       -> moveto.loc1
...
```

Reactive plans

- condition-**action** rules over an alphabet of plan actions

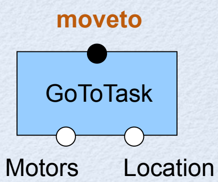
Includes alternative paths to the goals if there are unpredicted environment changes



component assembly

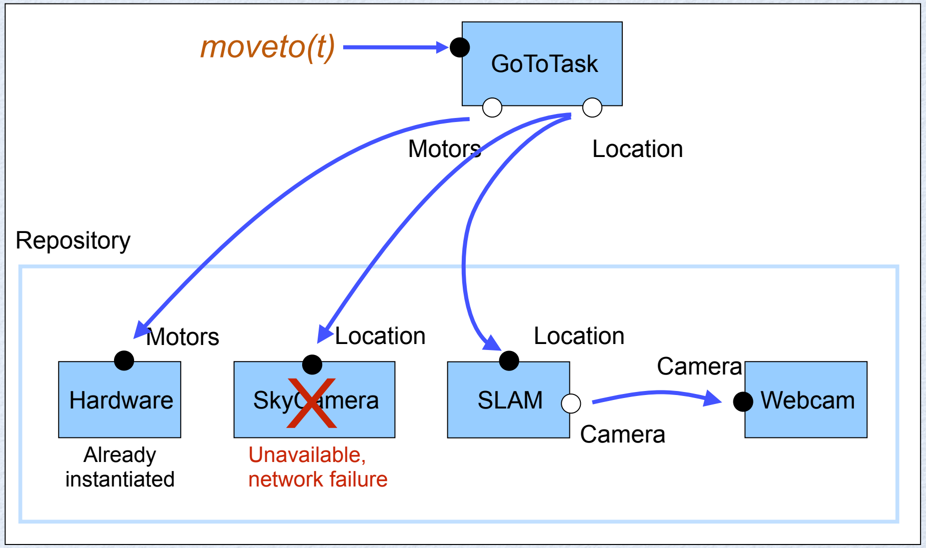
Derive configurations by mapping plan actions to components :

- primitive plan actions (pickup, moveto,…) are associated with the provided services of components which the plan interpreter can call
- elaborate and assemble components using dependencies (required services)

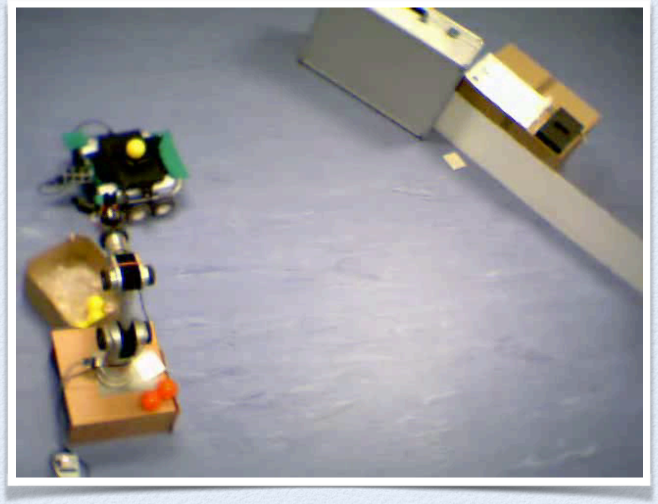


Mapping is a many to many relationship, providing alternatives

component assembly

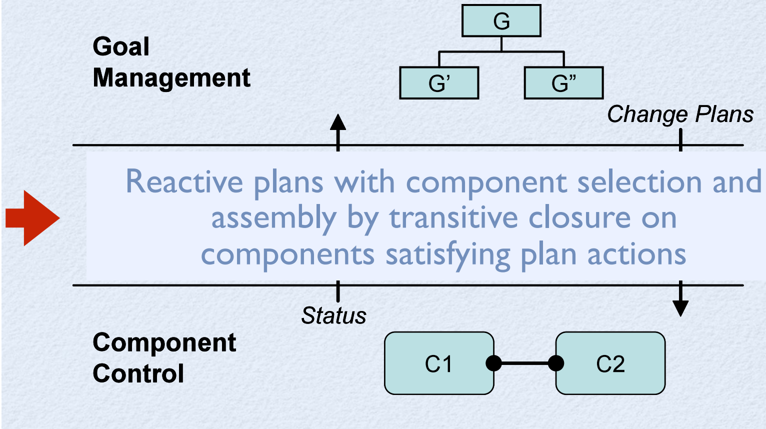


adaptation demonstration



Adaptation
 may
 require
 component
 reselection
or
alternative
 plan
 selection
or
replanning

three layer architecture



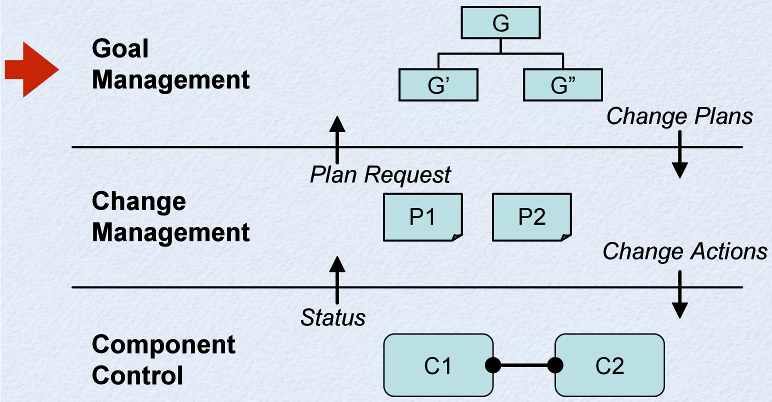
1. Planning over abstract domain

2. Precomputed plans: component assembly and plan execution

3. Component execution and dynamic configuration

ICSE FOSE '07, SEAMS 2008, SEAMS 2011

three layer architecture



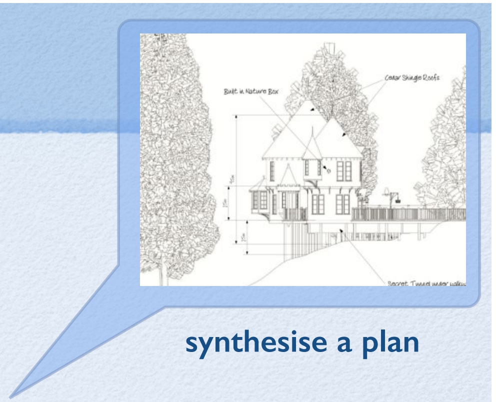
1. Planning over abstract domain

2. Precomputed plans: component assembly and plan execution

3. Component execution and dynamic configuration

ICSE FOSE '07, SEAMS 2008, SEAMS 2011

goal

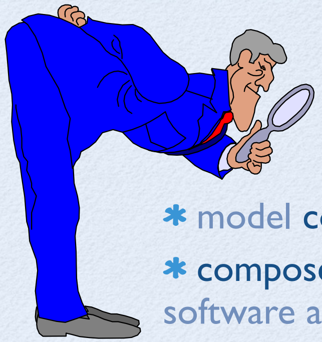


synthesise a plan

model-based planning

build a model

…earlier modelling research…



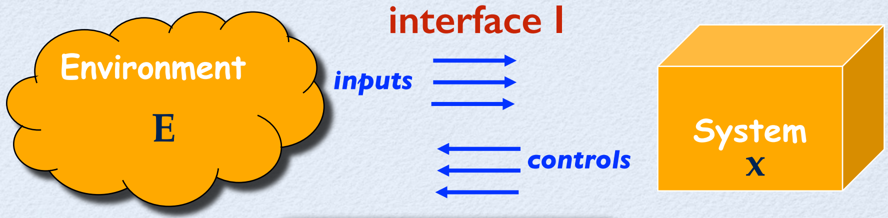
- ✱ model component behaviour as **LTS** in **FSP**
- ✱ compose behaviours according to the software architecture configuration

… model check properties using **LTSA**

ICSE '96, TOSEM '96, FSE '97, ESEC/FSE '99, book '99/2006

plan (controller) synthesis

Consider a plan as a winning strategy in an infinite two player game between the environment E and the system x with interface I such that goal G is always satisfied no matter what the order of inputs from environment.



|| composition of LTS

$$E \| x_I \models G$$

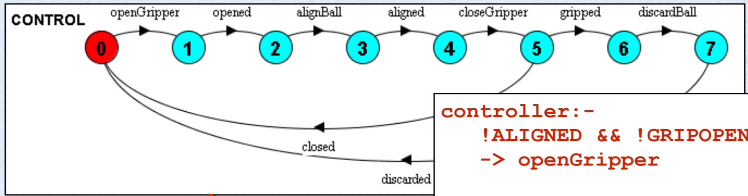
synthesise x

Goal G: Linear Temporal Logic property

Symbolic Controller Synthesis for Discrete and Timed Systems, Asarin, Maler & Pnueli, LNCS 999, 1995.

plan (controller) synthesis

Environment model (as || LTS)



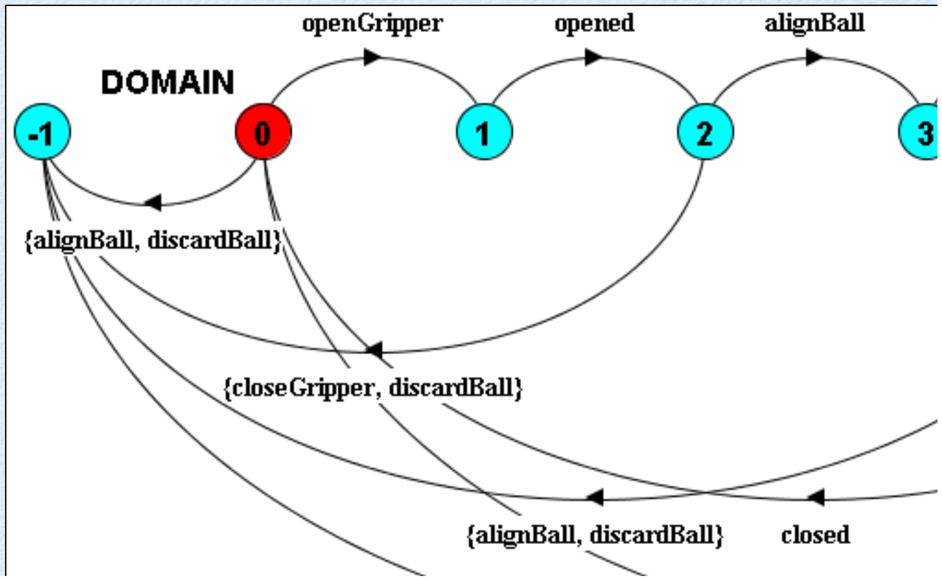

```
controller:-
   !ALIGNED && !GRIPOPEN && !PICKEDUP
   -> openGripper

   !ALIGNED && GRIPOPEN && !PICKEDUP
   -> alignBall

   !ALIGNED && !GRIPOPEN && PICKEDUP
   -> discardBall

   ALIGNED && GRIPOPEN && !PICKEDUP
   -> closeGripper
```

```
ltl_property SAFE4 =
    [](closeGripper -> ALIGNED)
ltl_property GETBALL =
    [](alignBall -> X closeGrippe
ltl_property PROGRESS =
    [](openGripper -> X alignBall)
```

Plan (as a controller)

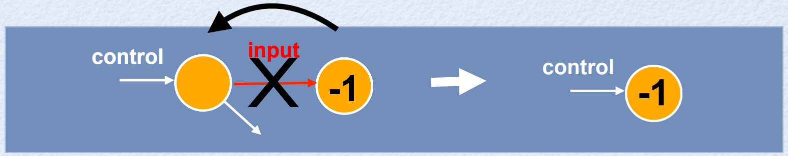
Goal specification (as LTL properties)

plan (controller) synthesis

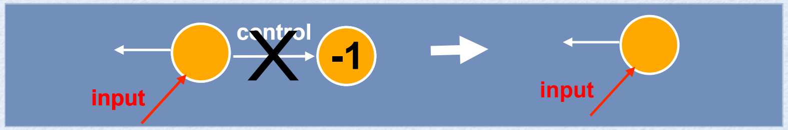


computing “winning” states

- By backward propagation of the **error state -1** for **inputs** from the environment:



- By removal of the **error state -1** for **controls** from the controller:

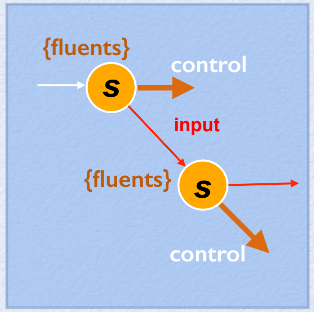


plan extraction

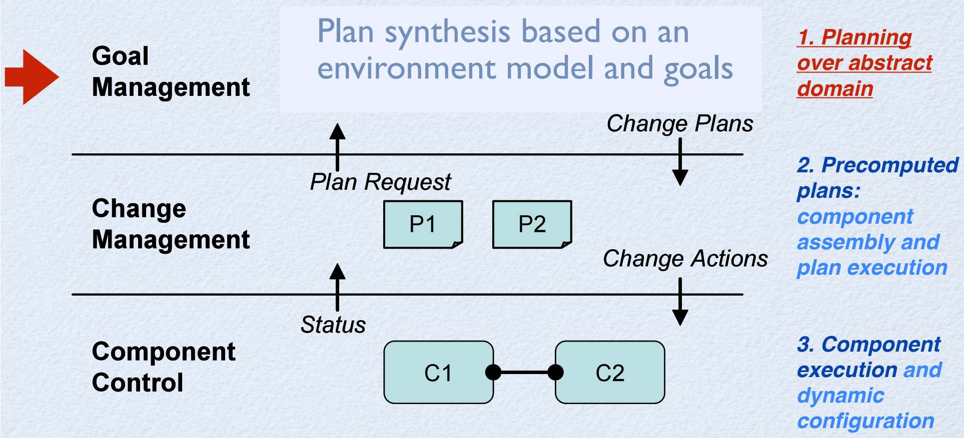
Reactive Plan computed from the control states **S**
(with outgoing transition labelled with control)

- Label states with fluent values
- Fluents form the preconditions for the control actions.

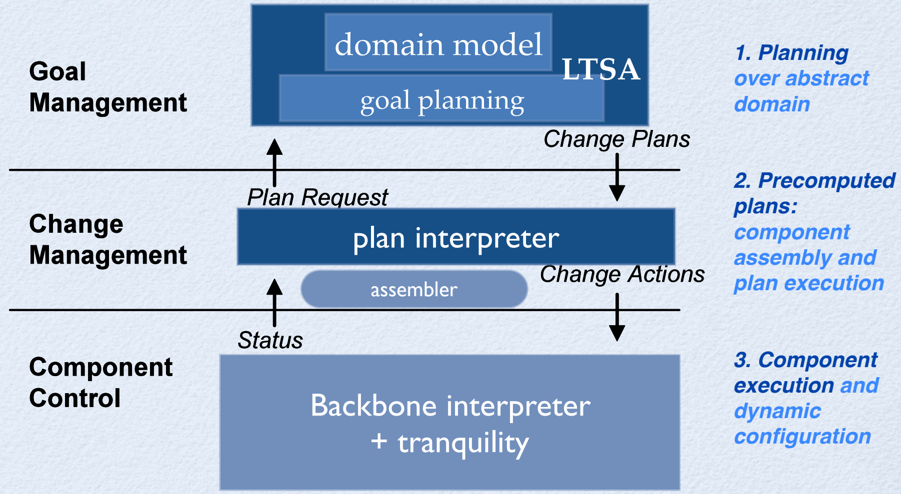
```
controller:-
   !ALIGNED && !GRIPOPEN && !PICKEDUP
   -> openGripper

   !ALIGNED && GRIPOPEN && !PICKEDUP
   -> alignBall

   !ALIGNED && !GRIPOPEN && PICKEDUP
   -> discardBall

   ALIGNED && GRIPOPEN && !PICKEDUP
   -> closeGripper
```



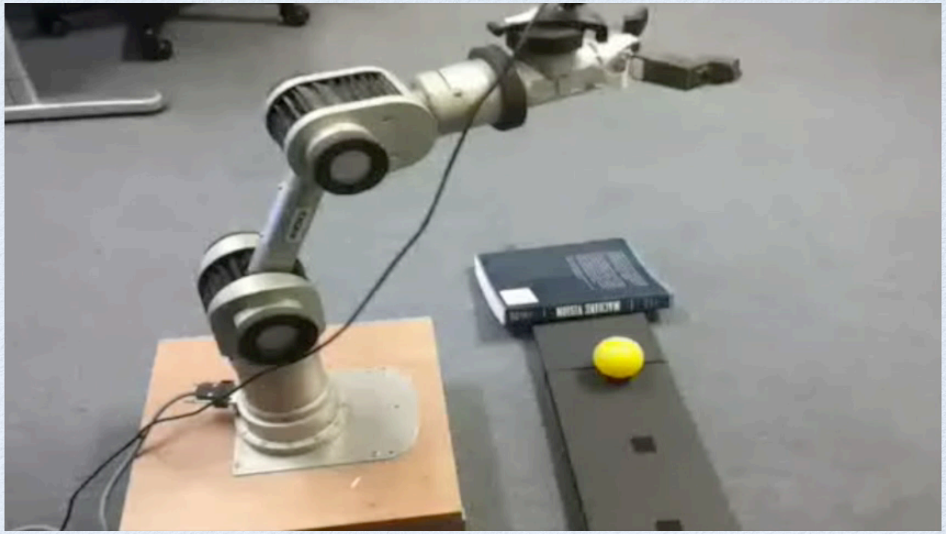
three layer architecture



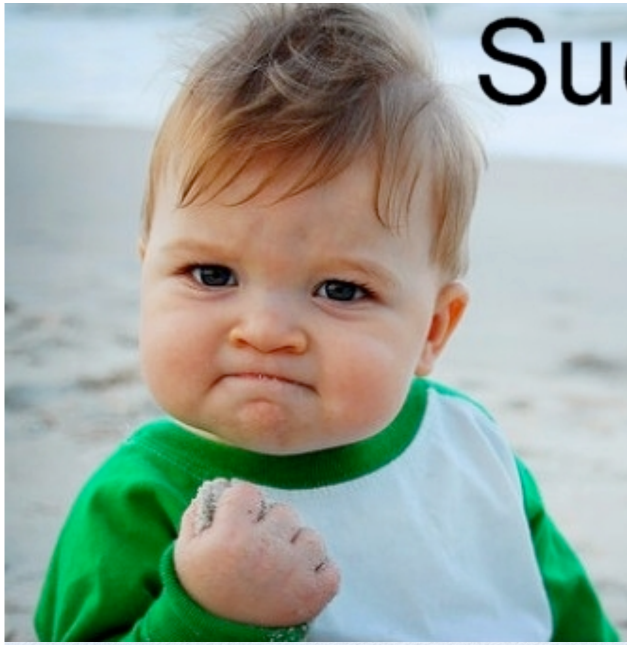
three layer architecture **realisation**



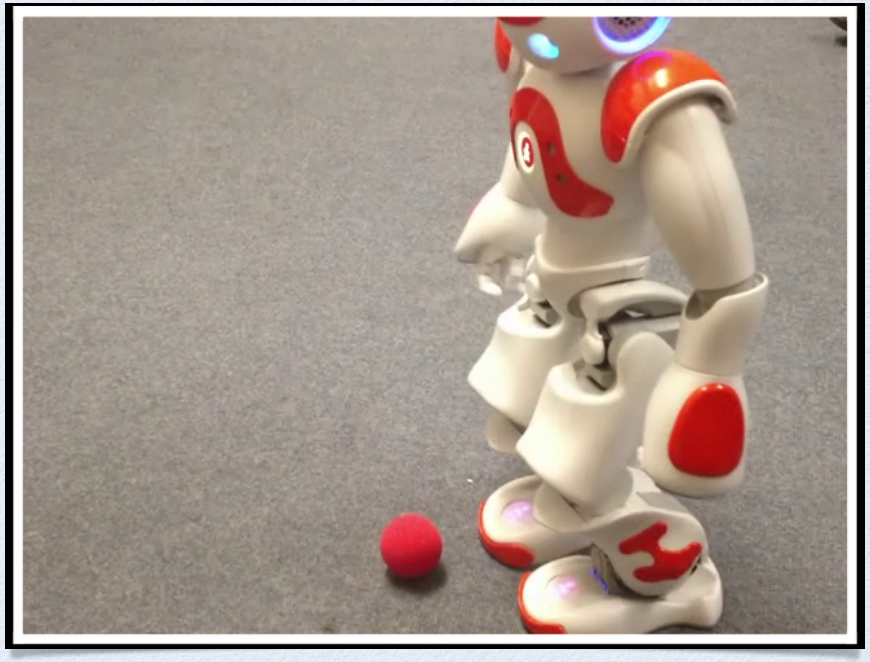
three layer architecture **realisation**



ICSE FOSE '07, SEAMS 2008, SEAMS 2011



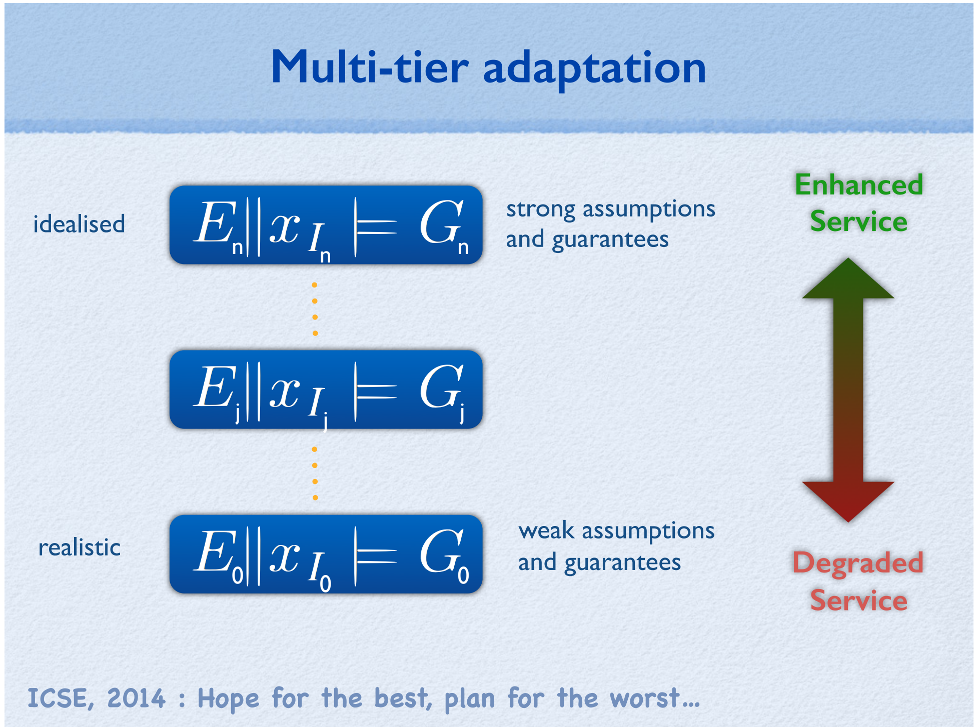
… mostly …



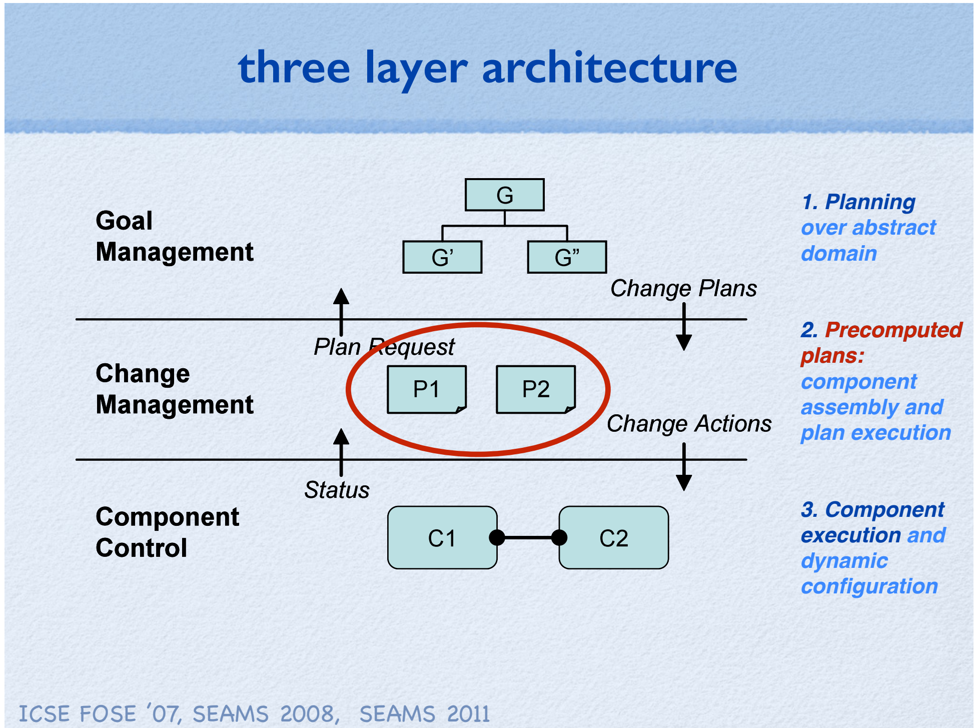
ICSE 2013 teaser demo

!

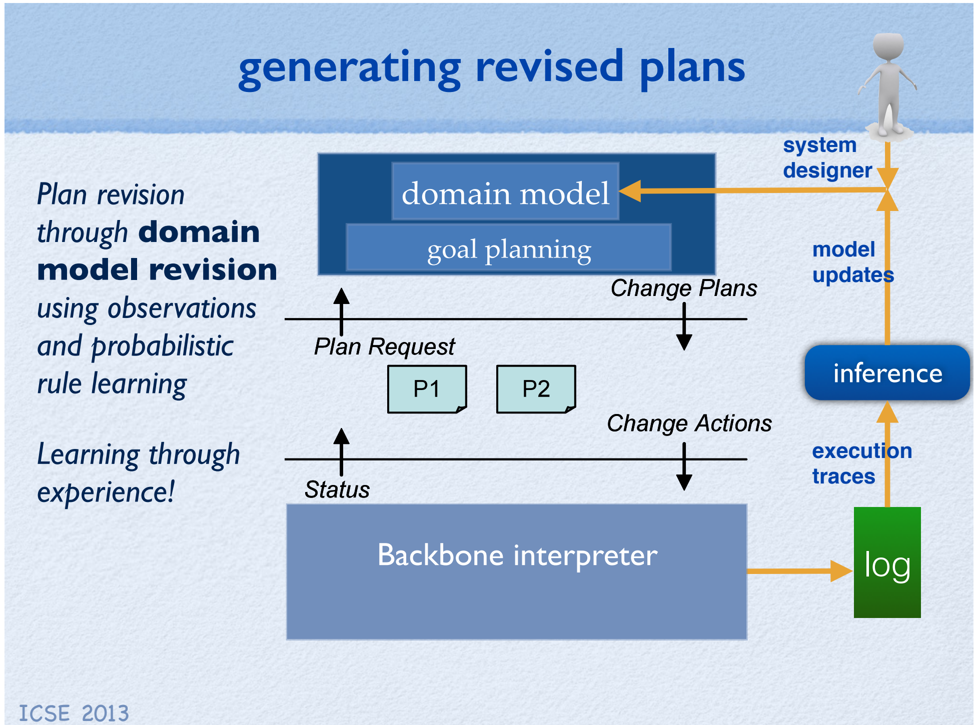
- shortcomings provide the challenges for further research …



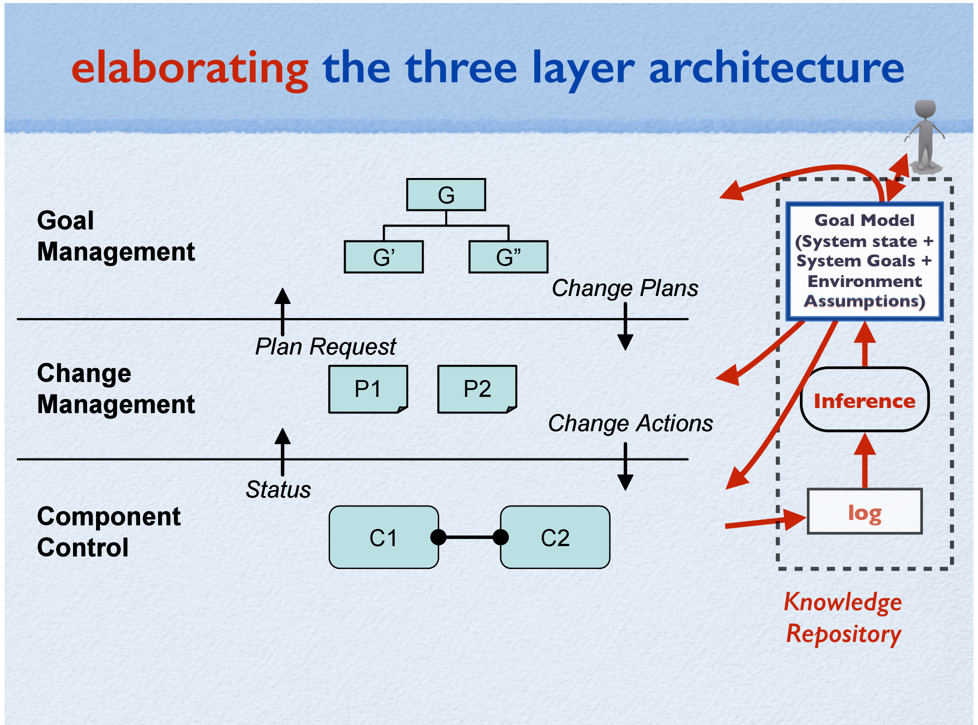
Multi-tier adaptation
idealised
$E_n \| x_{I_n} \models G_n$
strong assumptions and guarantees
Enhanced Service
$E_j \| x_{I_j} \models G_j$
realistic
$E_0 \| x_{I_0} \models G_0$
weak assumptions and guarantees
Degraded Service
ICSE, 2014 : Hope for the best, plan for the worst…



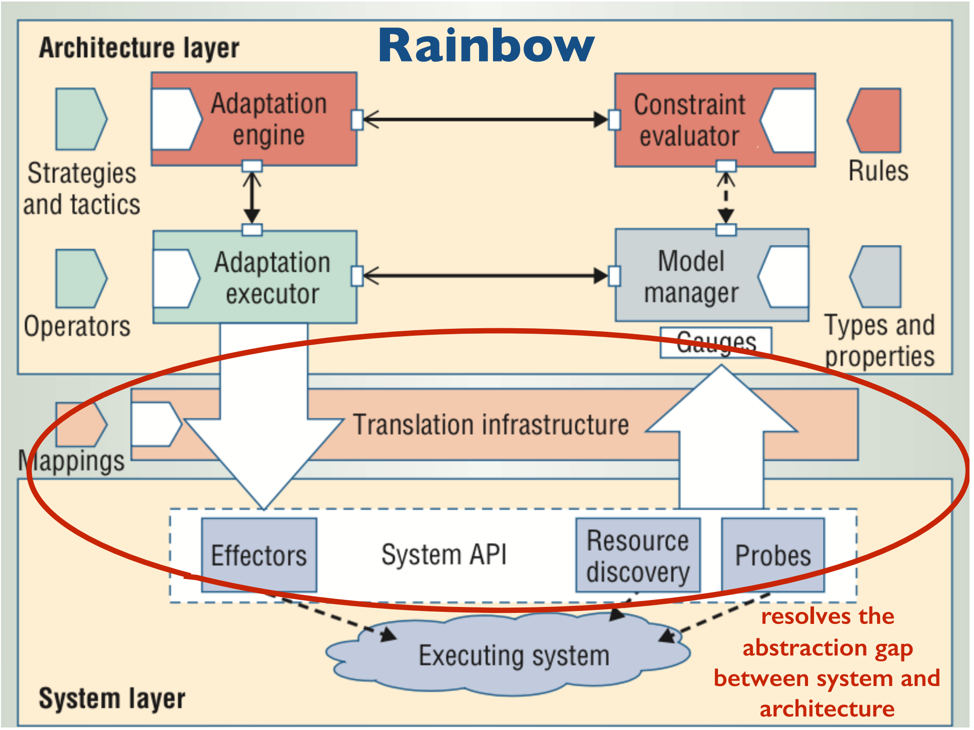
three layer architecture
Goal Management
G
G'
G"
Change Plans
Plan Request
Change Management
P1
P2
Change Actions
Status
Component Control
C1
C2
1. Planning over abstract domain
2. Precomputed plans: component assembly and plan execution
3. Component execution and dynamic configuration
ICSE FOSE '07, SEAMS 2008, SEAMS 2011

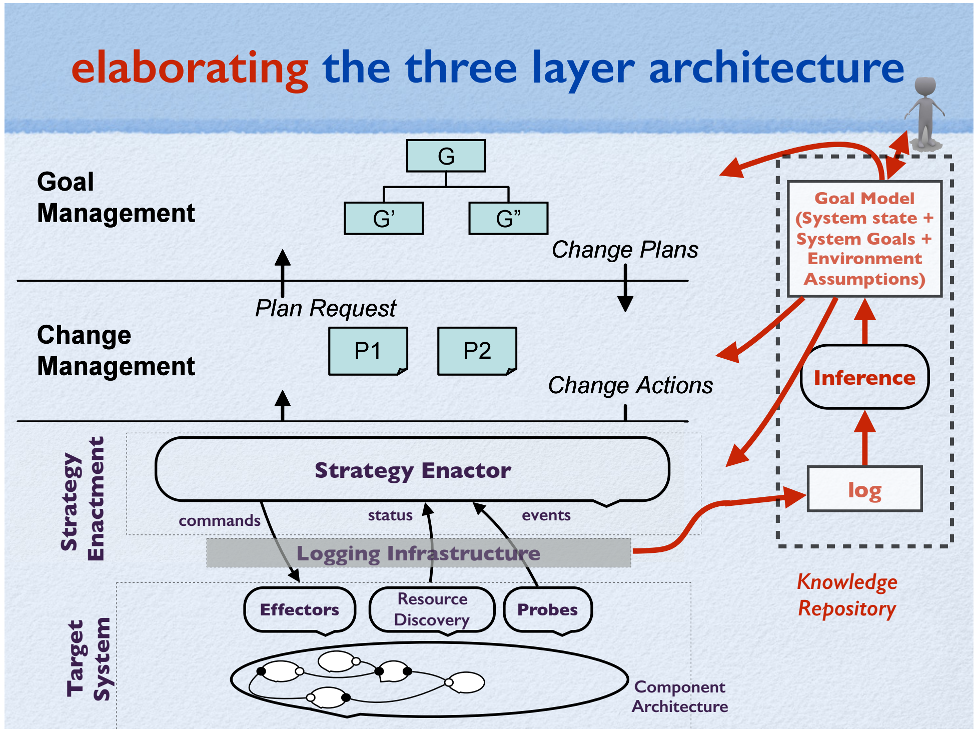


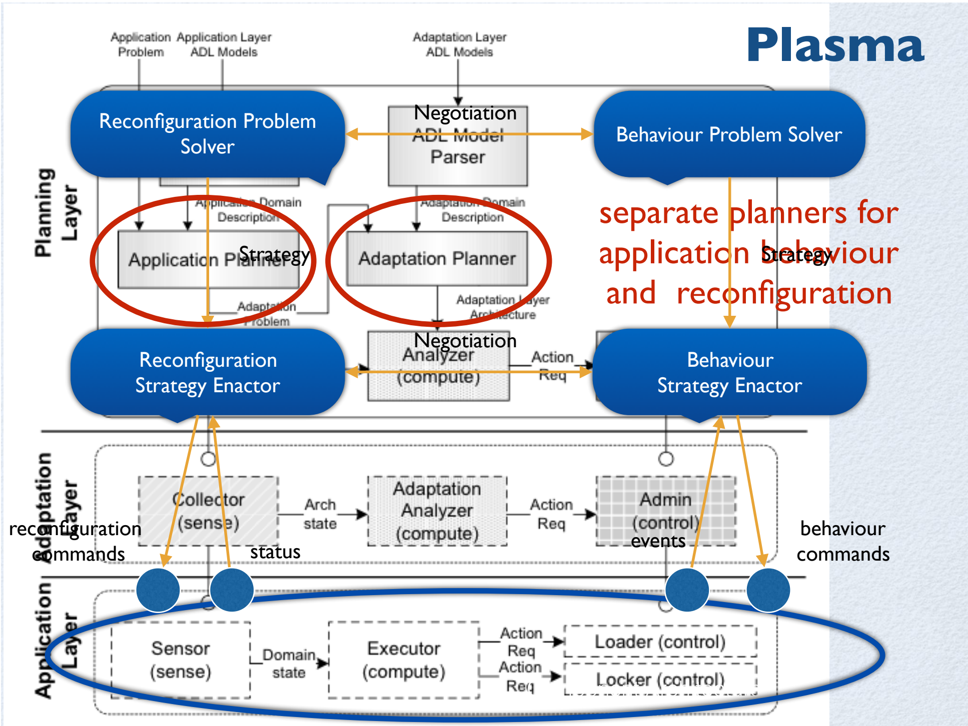
generating revised plans
Plan revision through domain model revision using observations and probabilistic rule learning
Learning through experience!
domain model
goal planning
system designer
model updates
Change Plans
Plan Request
P1
P2
Change Actions
inference
execution traces
Status
Backbone interpreter
log
ICSE 2013

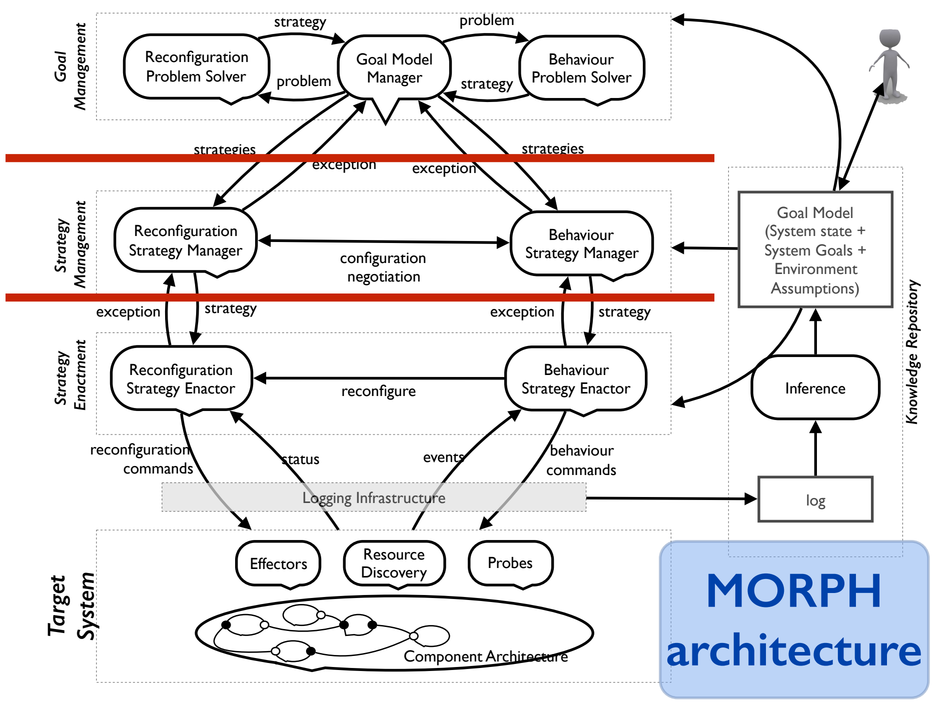


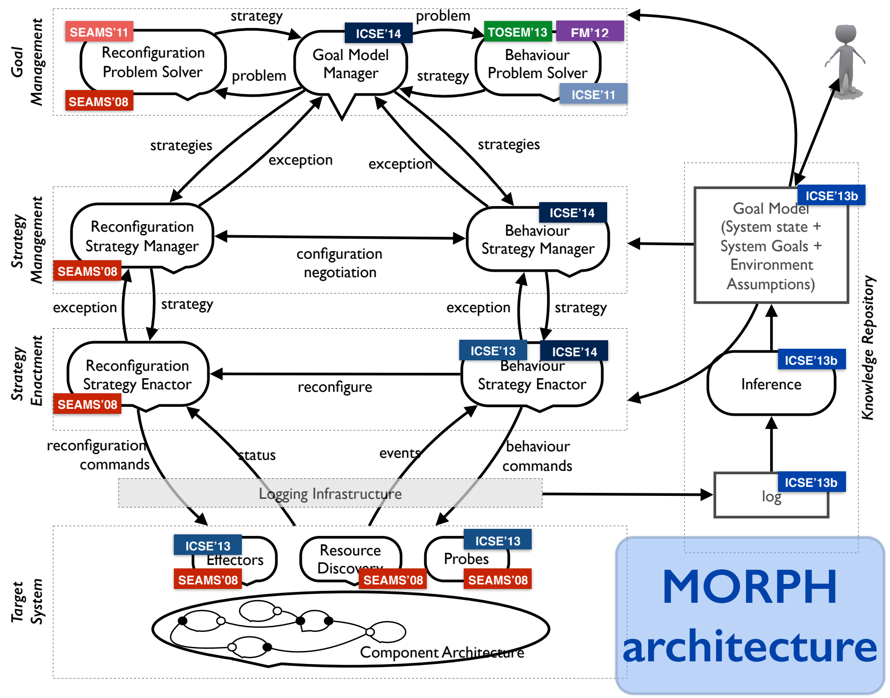
elaborating the three layer architecture
Goal Management
G
G'
G"
Change Plans
Plan Request
Change Management
P1
P2
Change Actions
Status
Component Control
C1
C2
Goal Model (System state + System Goals + Environment Assumptions)
Inference
log
Knowledge Repository


Rainbow
Architecture layer
Adaptation engine
Constraint evaluator
Strategies and tactics
Rules
Adaptation executor
Model manager
Operators
Types and properties
Gauges
Translation infrastructure
Mappings
Effectors
System API
Resource discovery
Probes
Executing system
System layer
resolves the abstraction gap between system and architecture


elaborating the three layer architecture
Goal Management
G
G'
G"
Change Plans
Plan Request
Change Management
P1
P2
Change Actions
Strategy Enactment
Strategy Enactor
commands
status
events
Logging Infrastructure
Target System
Effectors
Resource Discovery
Probes
Component Architecture
Goal Model (System state + System Goals + Environment Assumptions)
Inference
log
Knowledge Repository


Plasma
Application Problem
Application Layer ADL Models
Adaptation Layer ADL Models
Reconfiguration Problem Solver
Negotiation
ADL Model Parser
Behaviour Problem Solver
Planning Layer
Application Domain Description
Adaptation Domain Description
Application Planner
Strategy
Adaptation Planner
separate planners for application behaviour and reconfiguration
Adaptation Problem
Adaptation Layer Architecture
Reconfiguration Strategy Enactor
Negotiation
Analyzer (compute)
Action Req
Behaviour Strategy Enactor
Adaptation Layer
Collector (sense)
Arch state
Adaptation Analyzer (compute)
Action Req
Admin (control)
events
reconfiguration commands
status
behaviour commands
Application Layer
Sensor (sense)
Domain state
Executor (compute)
Action Req
Action Req
Loader (control)
Locker (control)


Goal Management
strategy
problem
Reconfiguration Problem Solver
problem
Goal Model Manager
strategy
Behaviour Problem Solver
strategies
exception
exception
strategies
Strategy Management
Reconfiguration Strategy Manager
configuration negotiation
Behaviour Strategy Manager
Goal Model (System state + System Goals + Environment Assumptions)
exception
strategy
exception
strategy
Strategy Enactment
Reconfiguration Strategy Enactor
reconfigure
Behaviour Strategy Enactor
Inference
Knowledge Repository
reconfiguration commands
status
events
behaviour commands
Logging Infrastructure
log
Target System
Effectors
Resource Discovery
Probes
Component Architecture
MORPH architecture



MORPH architecture

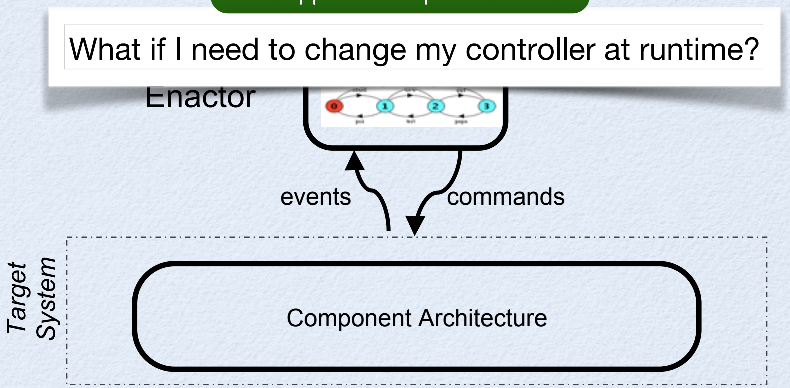
our architectural vision

Provide a reference architecture which …

- accommodates specific research aspects more clearly
- facilitates evaluation, validation and comparison of specific approaches
- provides a pick-and-mix (plug-and-play) architecture

Dynamic Controller Update

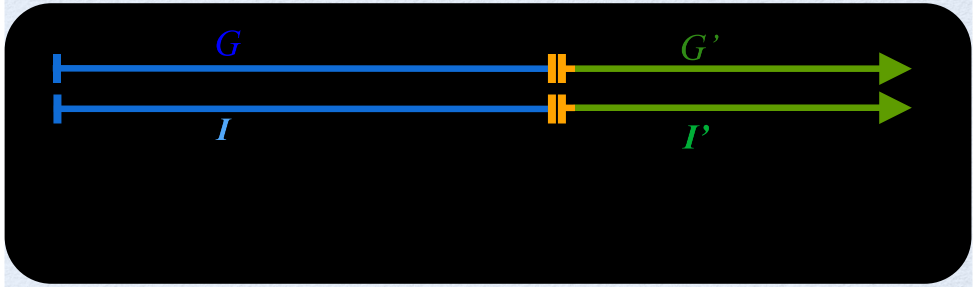
$$E' \| x'_{I'} \models G'$$

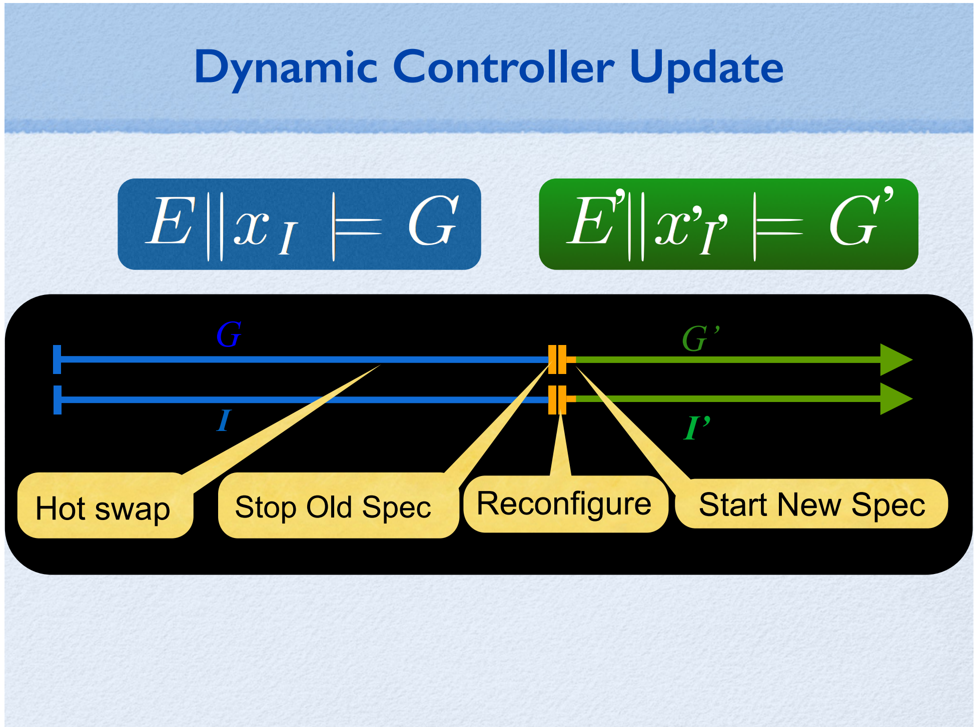


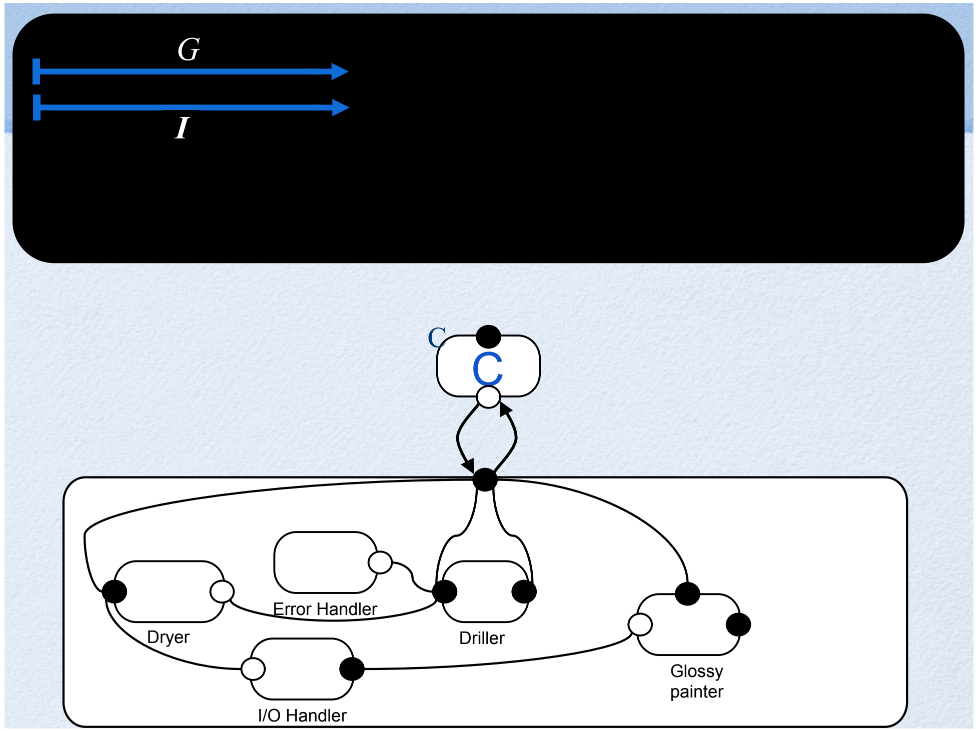
Dynamic Controller Update

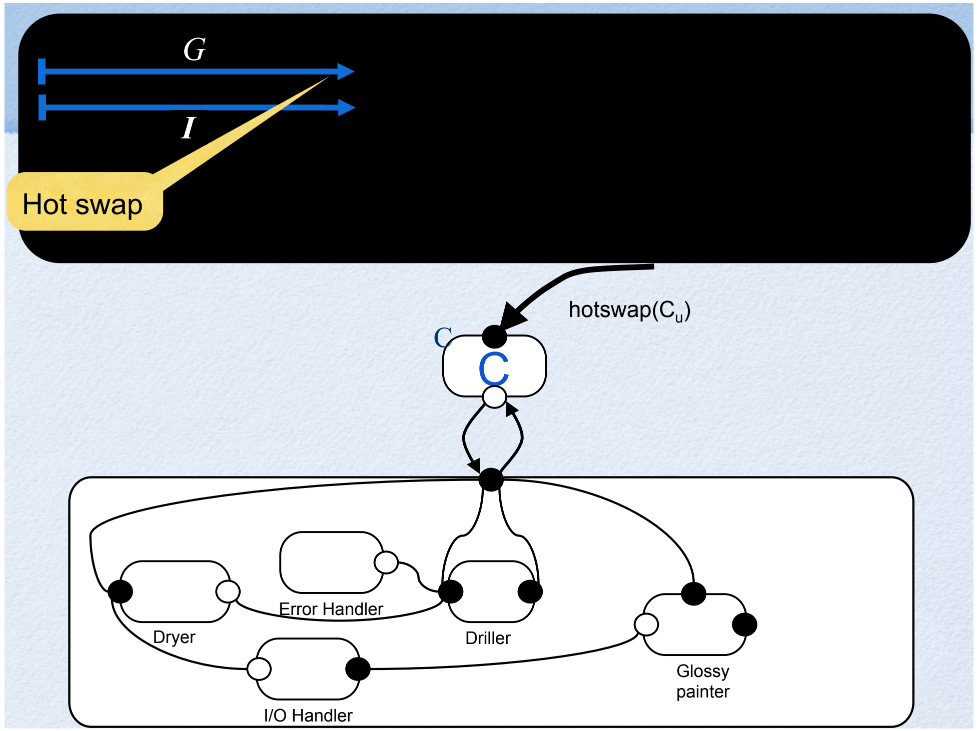
$$E \| x_I \models G$$

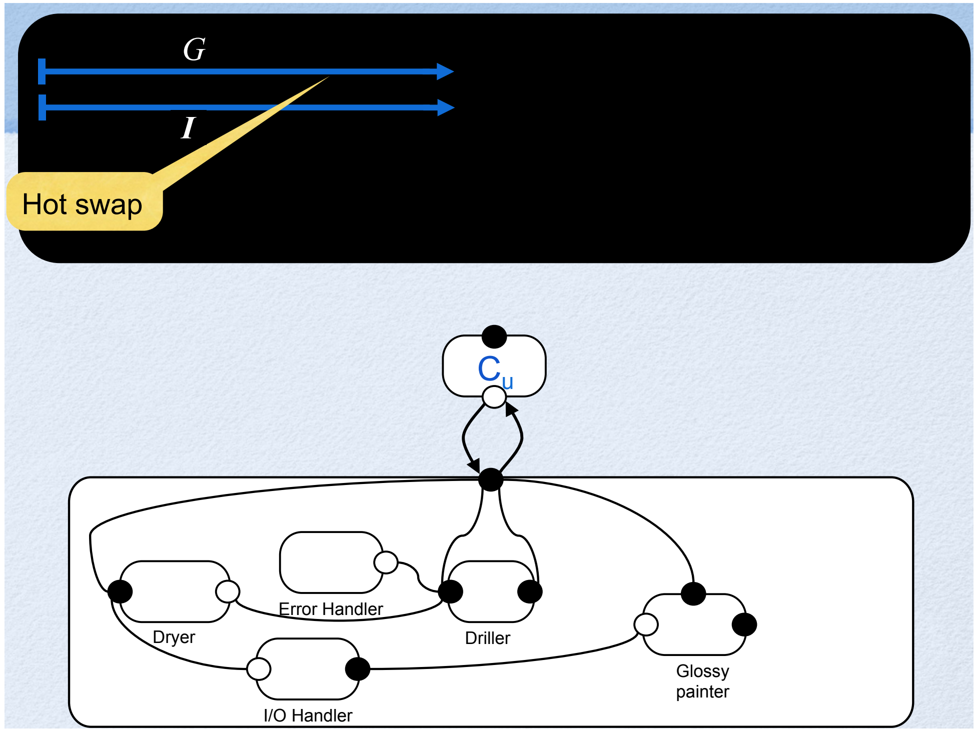
$$E' \| x'_{I'} \models G'$$

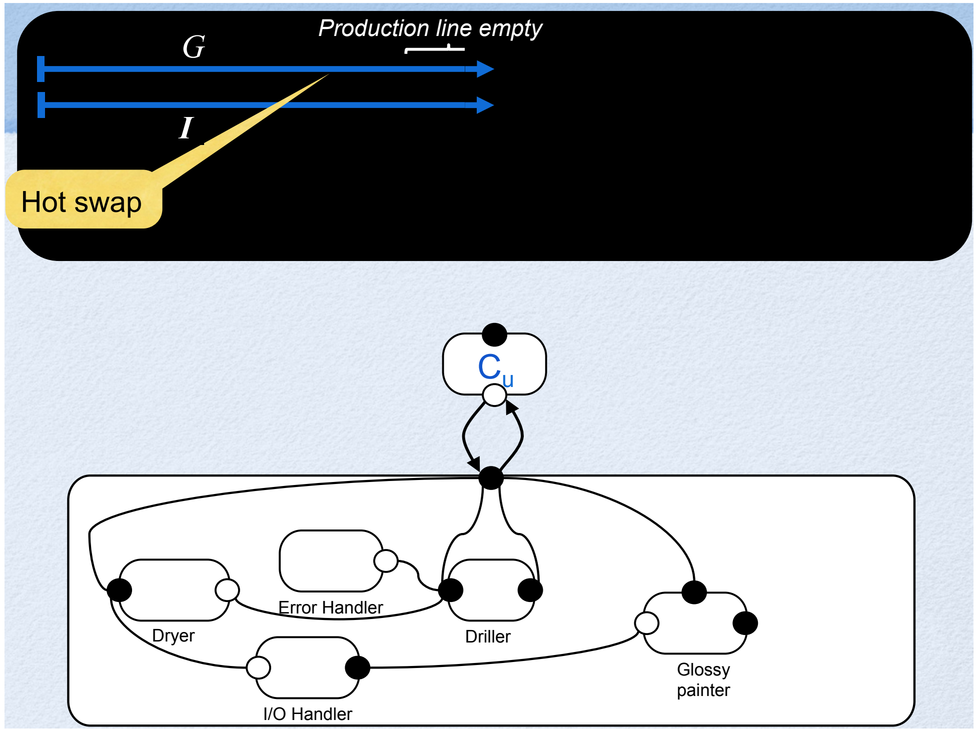


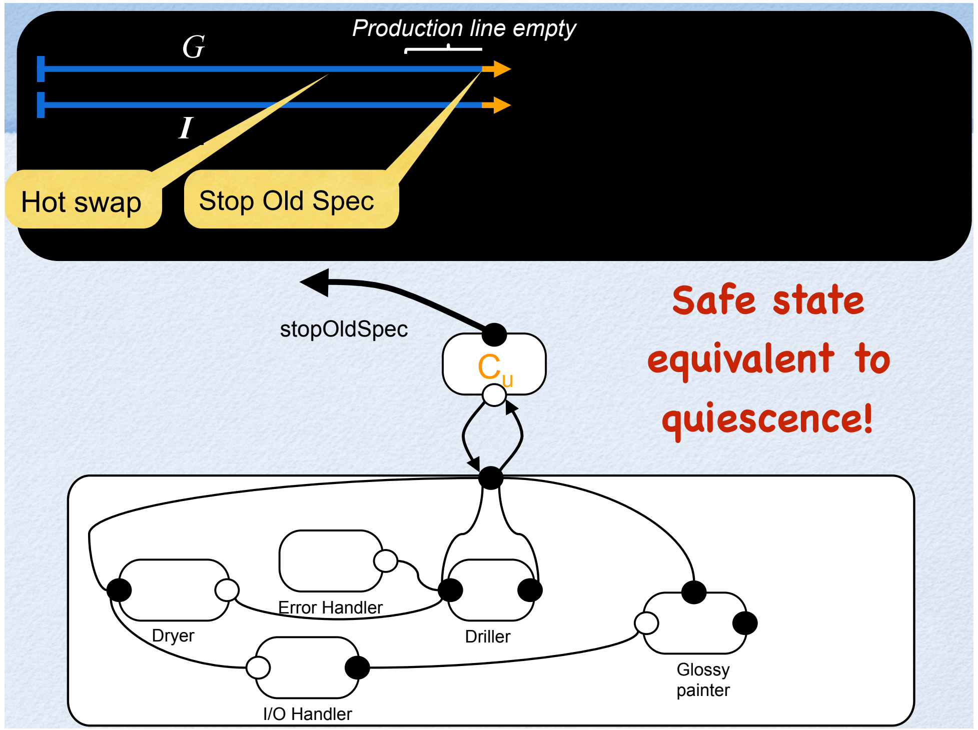

Dynamic Controller Update
$E \| x_I \models G$
$E' \| x'_{I'} \models G'$
G
I
G'
I'
Hot swap
Stop Old Spec
Reconfigure
Start New Spec

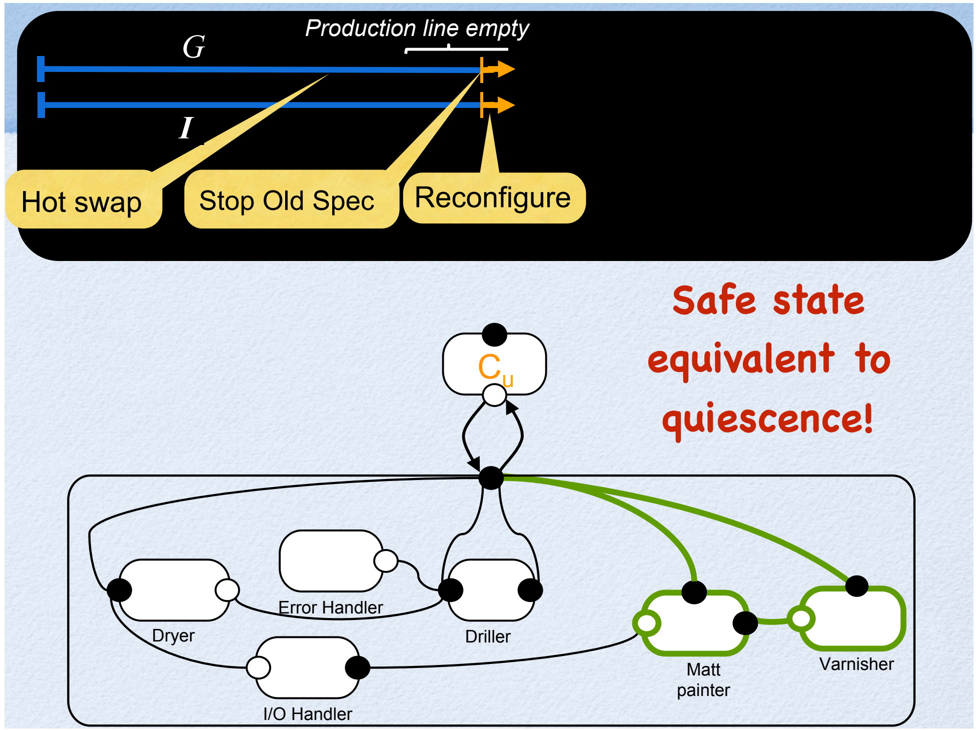

G
I
C
C
Error Handler
Dryer
Driller
Glossy
painter
I/O Handler

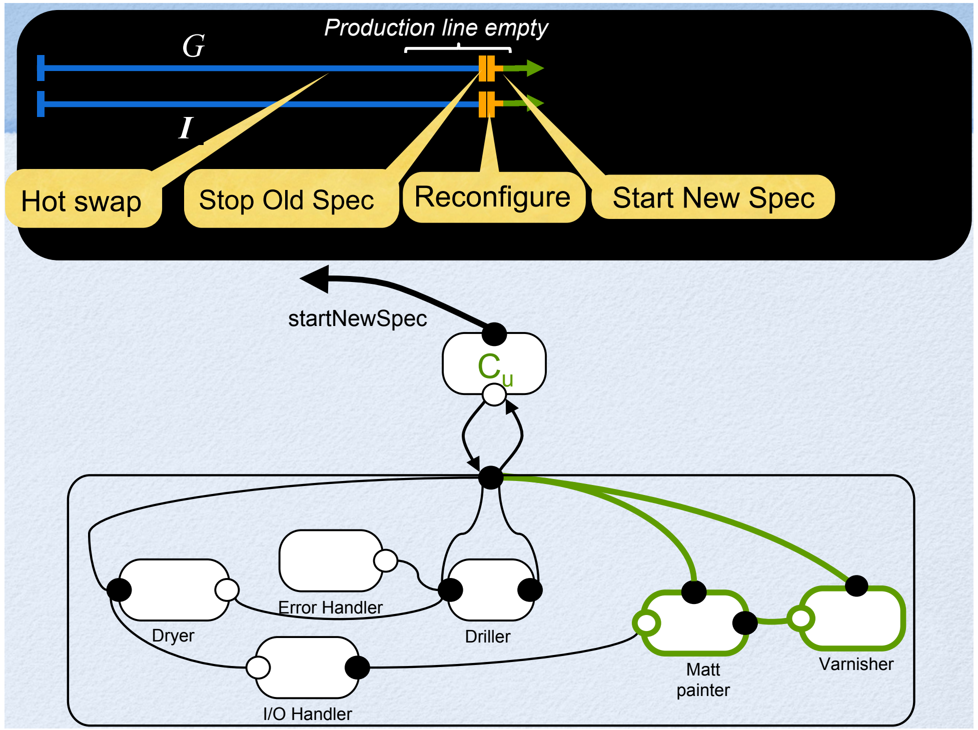

G
I
Hot swap
hotswap(C_u)
C
C
Error Handler
Dryer
Driller
Glossy
painter
I/O Handler

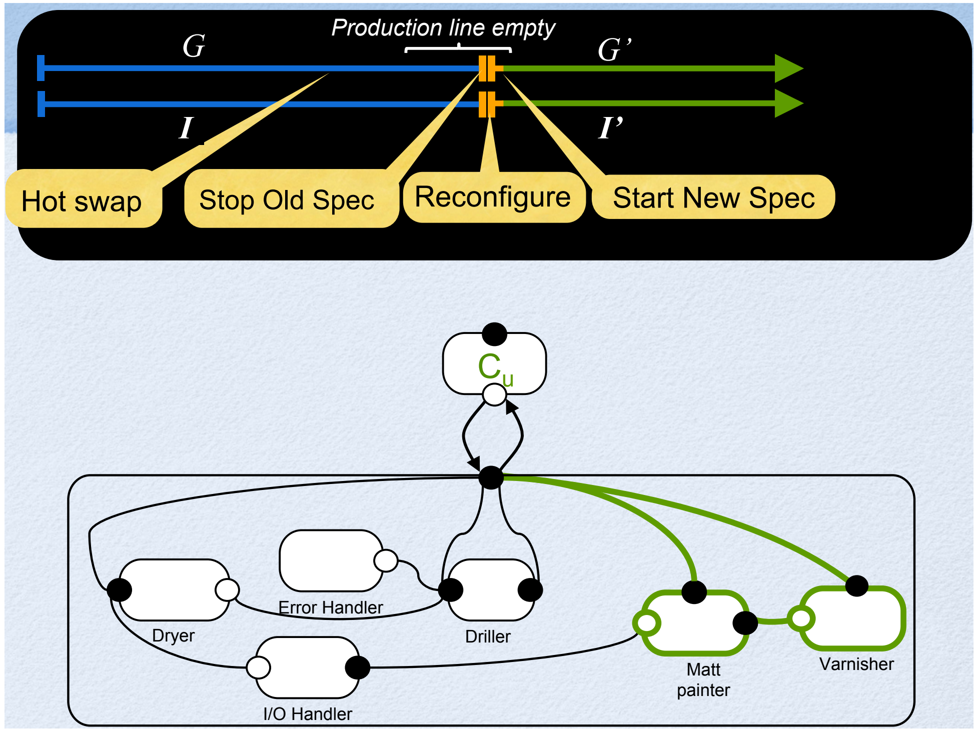

G
I
Hot swap
C_u
Error Handler
Dryer
Driller
Glossy
painter
I/O Handler

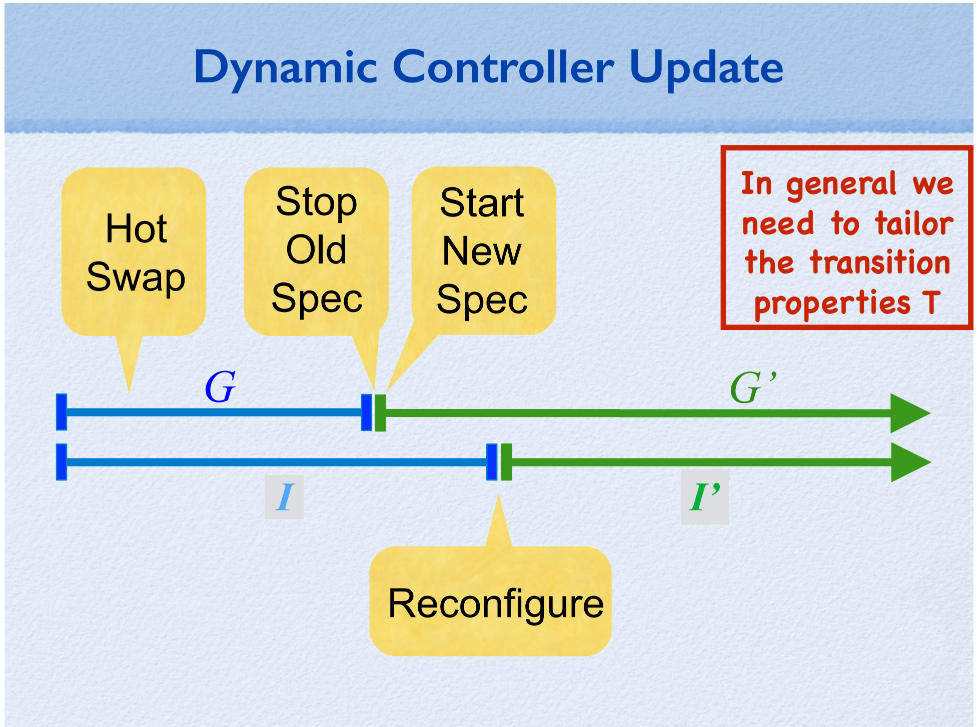

Production line empty
G
I
Hot swap
C
u
Error Handler
Dryer
Driller
Glossy
painter
I/O Handler

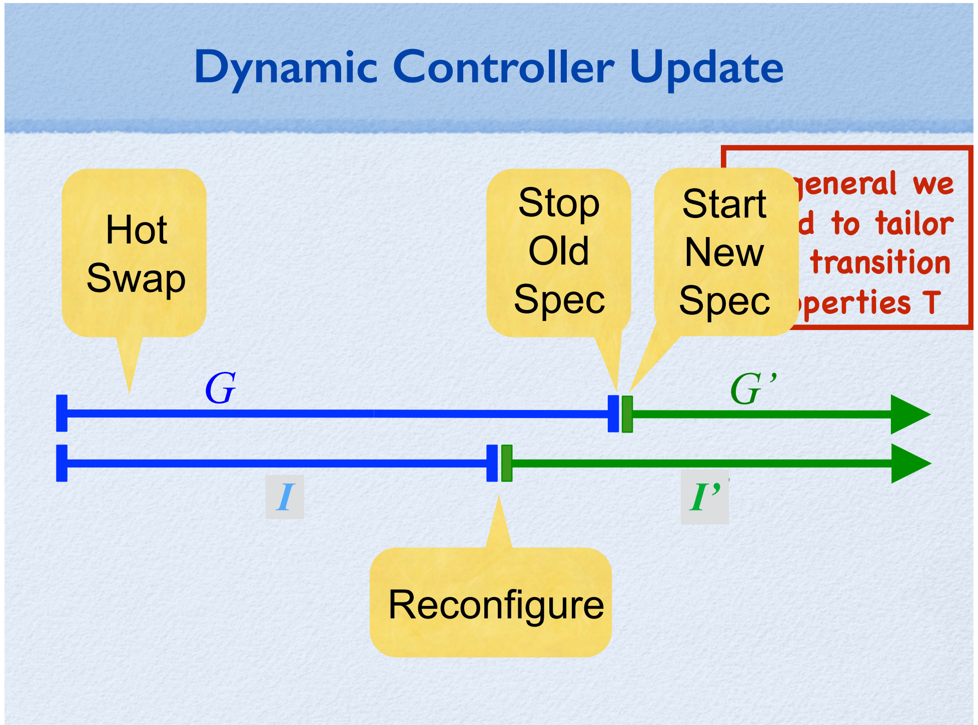

Production line empty
G
I
Hot swap
Stop Old Spec
stopOldSpec
Safe state
equivalent to
quiescence!
C
u
Error Handler
Dryer
Driller
Glossy
painter
I/O Handler

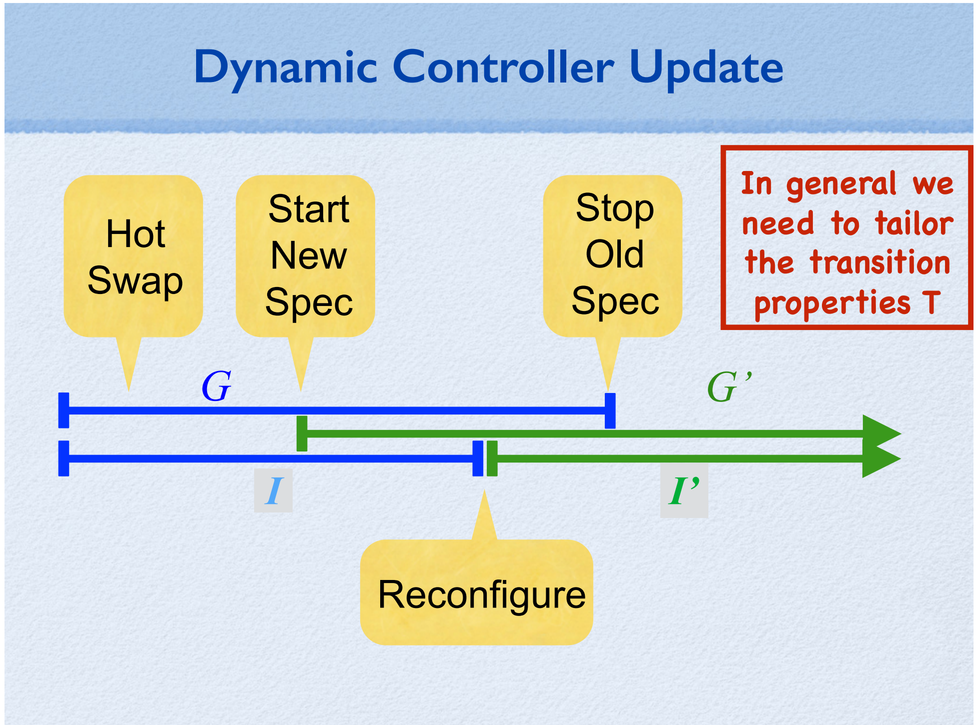

Production line empty
G
I
Hot swap
Stop Old Spec
Reconfigure
Safe state
equivalent to
quiescence!
C
u
Error Handler
Dryer
Driller
Matt
painter
Varnisher
I/O Handler


Production line empty
G
I
Hot swap
Stop Old Spec
Reconfigure
Start New Spec
startNewSpec
C
u
Error Handler
Dryer
Driller
Matt
painter
Varnisher
I/O Handler

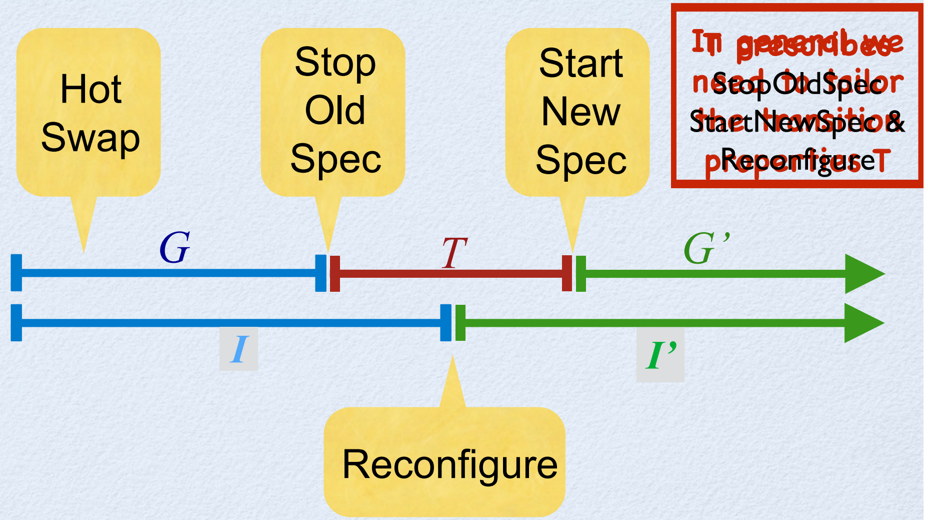

Production line empty
G
G'
I
I'
Hot swap
Stop Old Spec
Reconfigure
Start New Spec
C_u
Error Handler
Dryer
Driller
Matt painter
Varnisher
I/O Handler


Dynamic Controller Update
Hot Swap
Stop Old Spec
Start New Spec
In general we need to tailor the transition properties T
G
G'
I
I'
Reconfigure

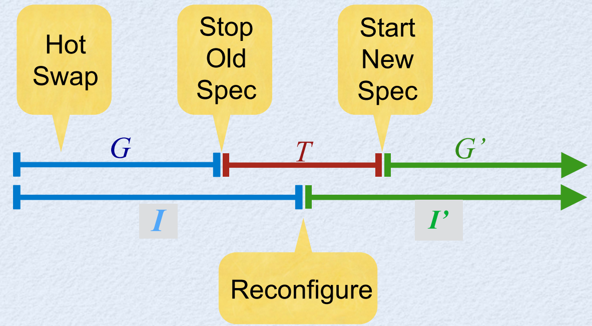

Dynamic Controller Update
Hot Swap
Stop Old Spec
Start New Spec
general we
d to tailor
transition
perties T
G
G'
I
I'
Reconfigure


Dynamic Controller Update
Hot Swap
Start New Spec
Stop Old Spec
In general we need to tailor the transition properties T
G
G'
I
I'
Reconfigure

Dynamic Controller Update

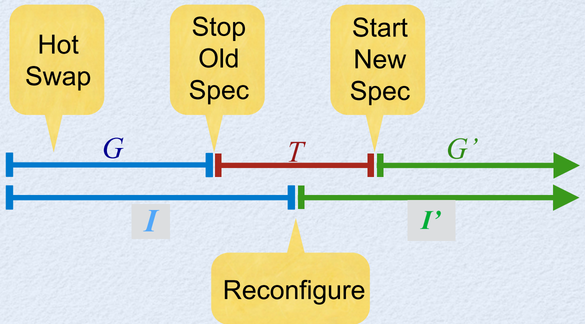


Dynamic Controller Specification



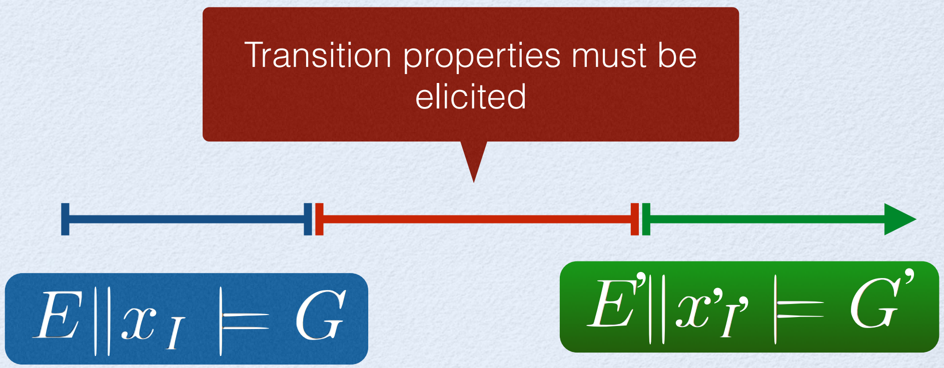
- G holds until StopOldSpec
- T holds
- G' holds after StartNewSpec
- If HotSwap then StartOldSpec, StartNewSpec and Reconfigure will occur

Dynamic Controller Specification

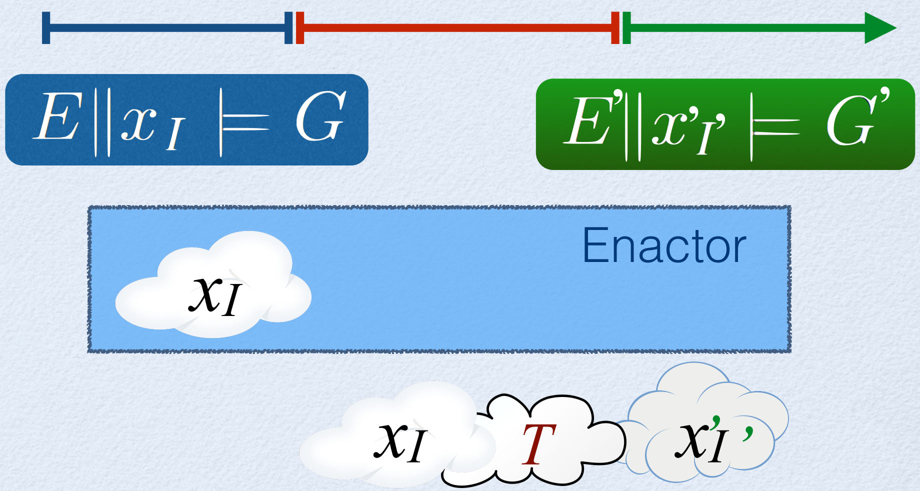


1. $G\ \mathbf{W}\ stopOldSpec$
2. T
3. $\square(startNewSpec \implies \square G')$
4. $\square(hotSwap \implies (\lozenge stopOldSpec \wedge \lozenge reconfigure \wedge \lozenge startNewSpec))$

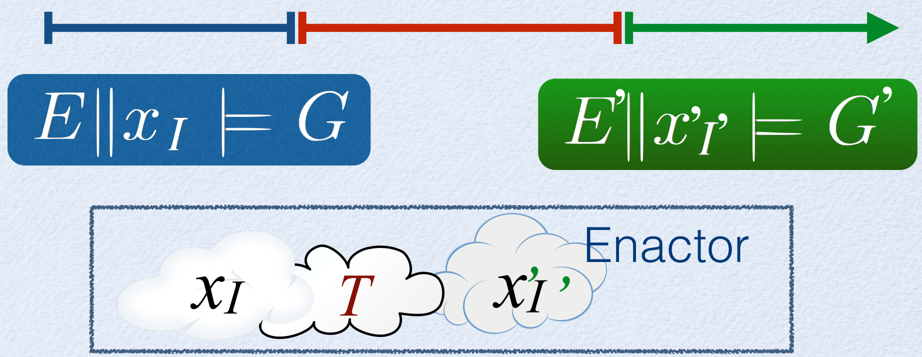
Dynamic Controller Transition



Dynamic Controller Hotswap



Dynamic Controller Hotswap



Dynamic Controller Update

- **General:** Supports explicit transition requirements and reconfiguration
- **Assured:** System is guaranteed to reach an updatable state
- **Correct**: Transition requirements and new specification are guaranteed by construction
- **Fully automated:** We use controller synthesis

in conclusion ...

Self-Managed Adaptive Systems

…. the challenges of **change** …

environment
goals
capabilities

models @ runtime

…. to automate and run on-line what is currently off-line!

and …

… need to use rigorous techniques and formal methods

SEFM

… an appropriate **architecture**

a sound foundation and context for research.

Bliss