Adventures in Adaptation:

a software engineering playground!

Jeff Kramer

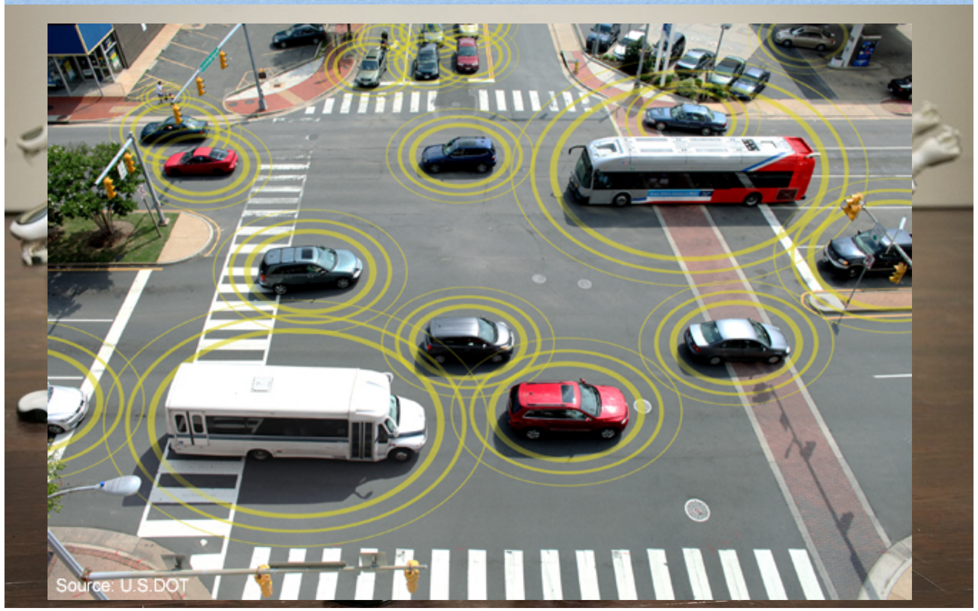
Imperial College London

Adaptive and Self-Managed Systems

…. the challenge of *change* …

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

Adaptive and Self-Managed Systems

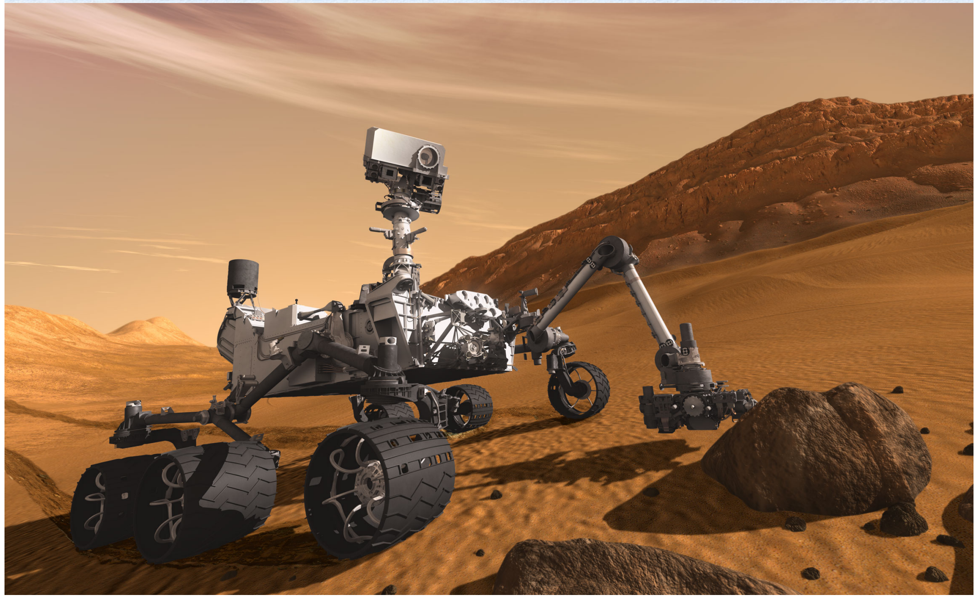


Adaptive and Self-Managed Systems

Adaptive *light* :
adjustment of runtime parameters in response to degraded performance or failure

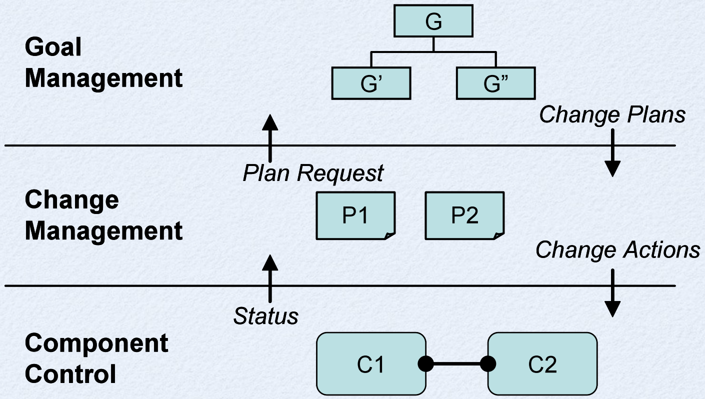
Adaptive *full fat* :
changes in functionality and performance in response to changes in the environment and/or goals

Adaptive and Self-Managed Systems



a software engineers' playground

three layer architecture



1. Planning *over abstract domain*

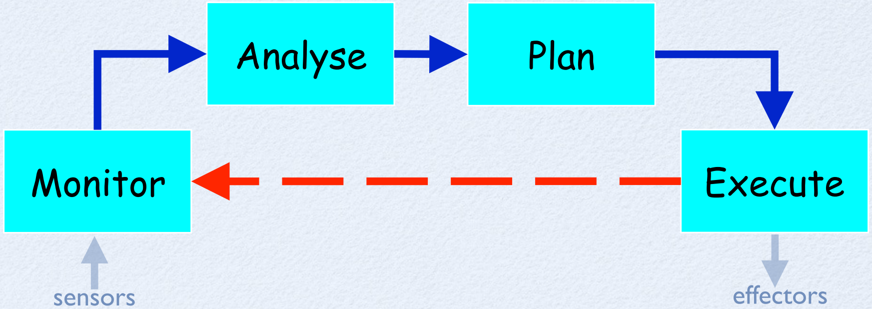
2. Precomputed plans: *component assembly and plan execution*

3. Component execution *and dynamic configuration*

?

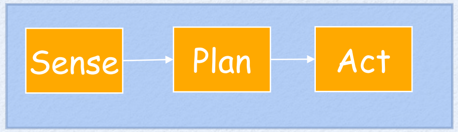
- why this architecture?
- how did we get here?
- where are we going?

MAPE cycle

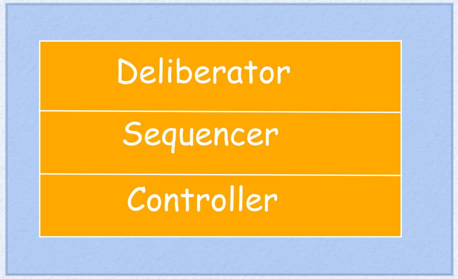


- a single feedback loop?
- response times?
- complexity?

inspiration from robotics



- 1970's



- 1998 (Gat)

1. Planning

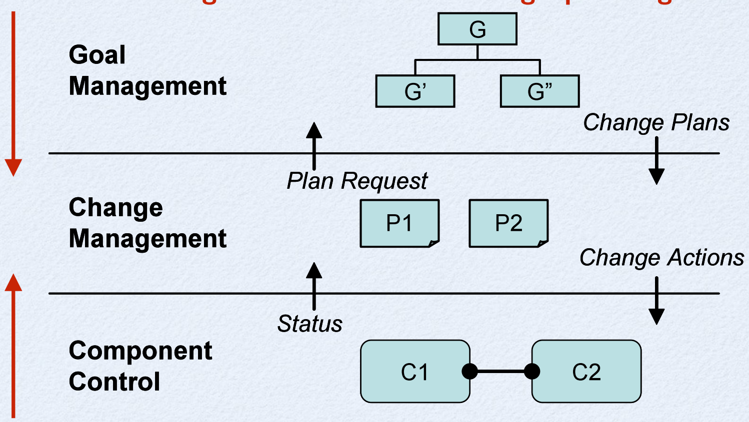
2. plan execution

3. component feedback control

- layering according to response times

three layer architecture

TD : decreasing statefullness and strategic planning



BU : increasing response time

***1. Planning** over abstract domain*

***2. Precomputed plans:** component assembly and plan execution*

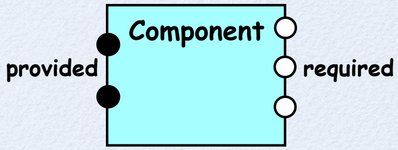
***3. Component execution** and dynamic configuration*

- **a separation of concerns**

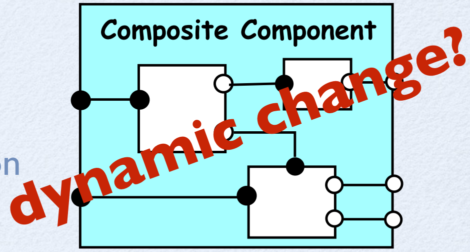
… some earlier research adventures …

CONIC and Darwin

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

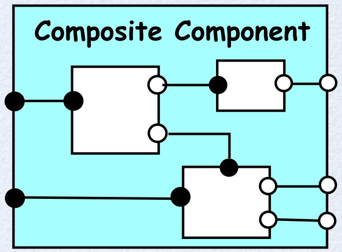


- an explicit configuration description (ADL)
  - third party instantiation and binding



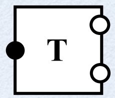
CONIC and Darwin

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

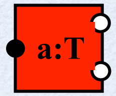


on-line dynamic change

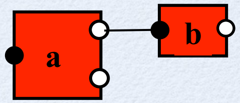
- **load** component type



- **create/delete** component instances



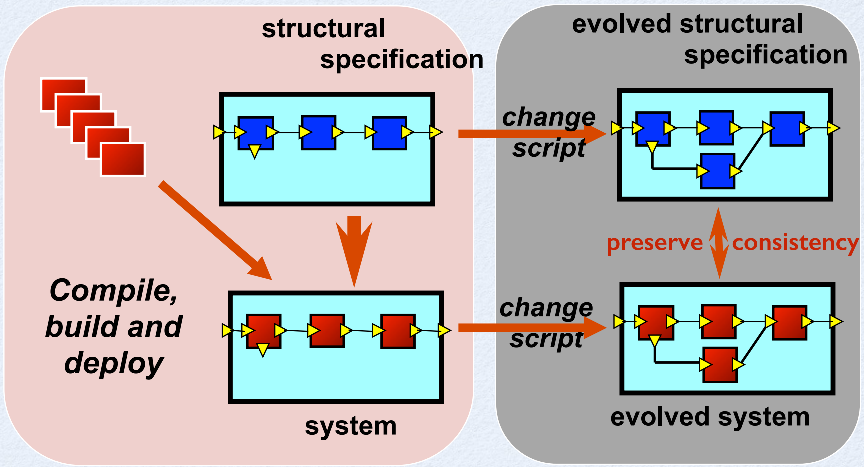
- **bind/unbind** component services



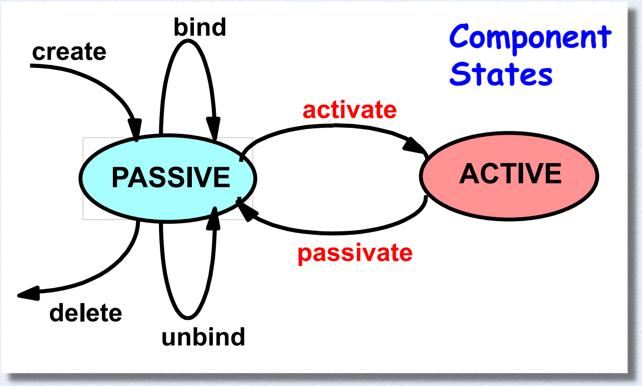
How can we do this **safely?**

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

configuration consistency



behaviour consistency



General change model: Separate the specification of *structural change* from the component *application behaviour.*

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

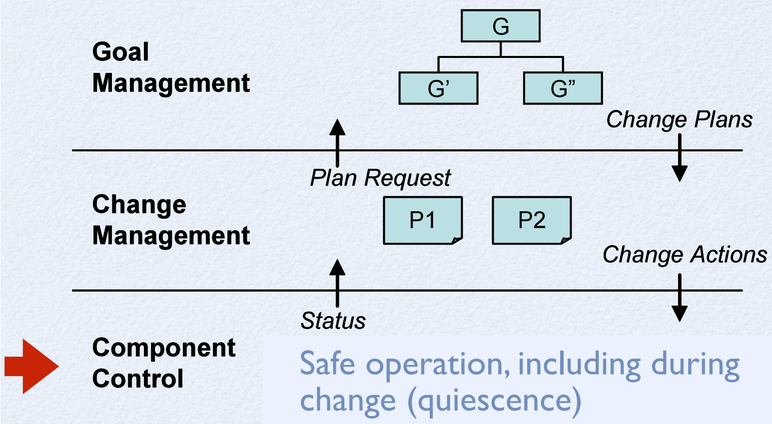
Quiescent : passive and no transactions will be initiated on it (ie. environment is passive)

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

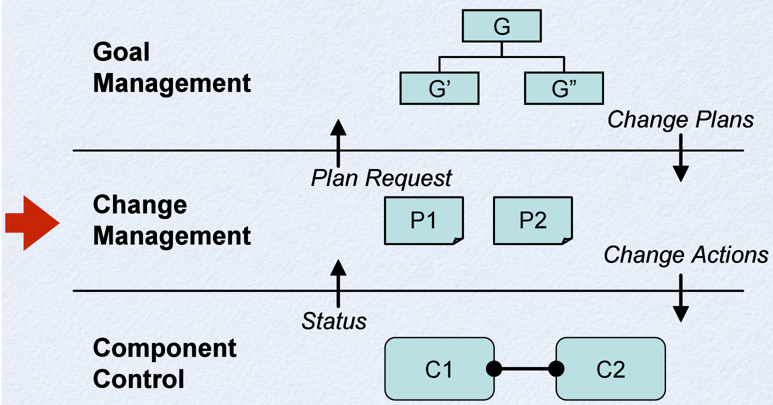


1. Planning over abstract domain

2. Precomputed plans: component assembly and plan execution

3. Component execution and dynamic configuration

three layer architecture

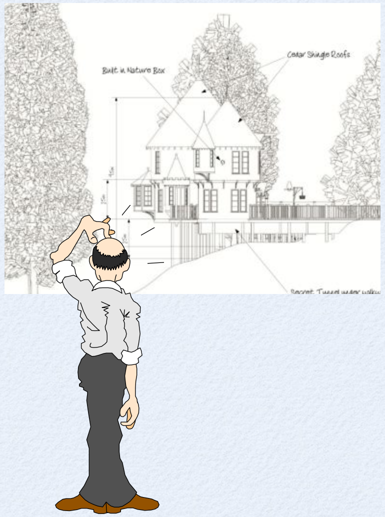


1. Planning over abstract domain

2. Precomputed plans: component assembly and plan execution

3. Component execution and dynamic configuration

component assembly?
plan execution?



plan execution

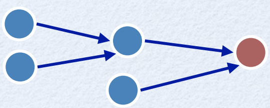
plan execution

```
...
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

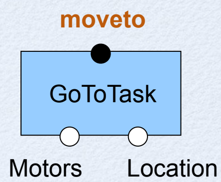
Includes alternative paths to the goals if there are unpredicted environment changes



component assembly

Derive configurations by mapping plan actions to components :

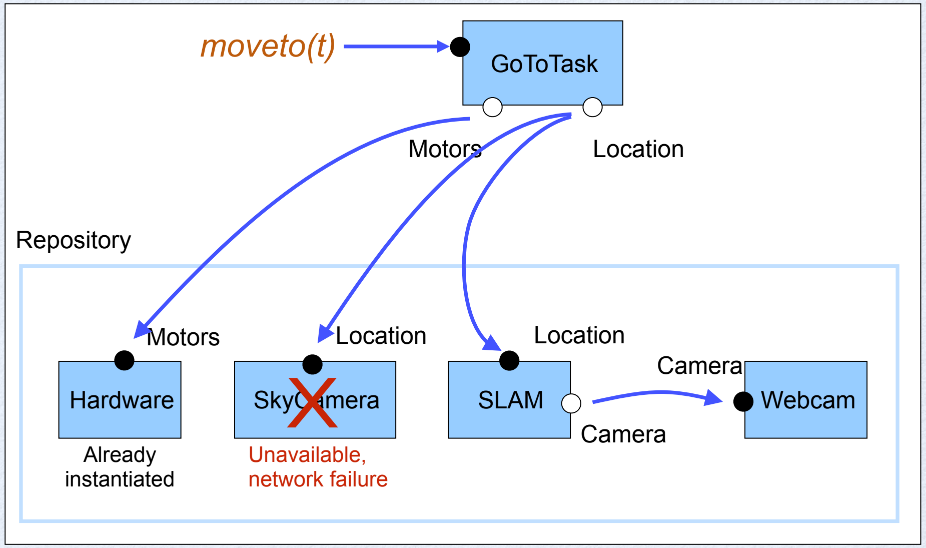
- primitive **plan actions** (pickup, moveto,…) are associated with the *provided services* of components which the plan interpreter can call



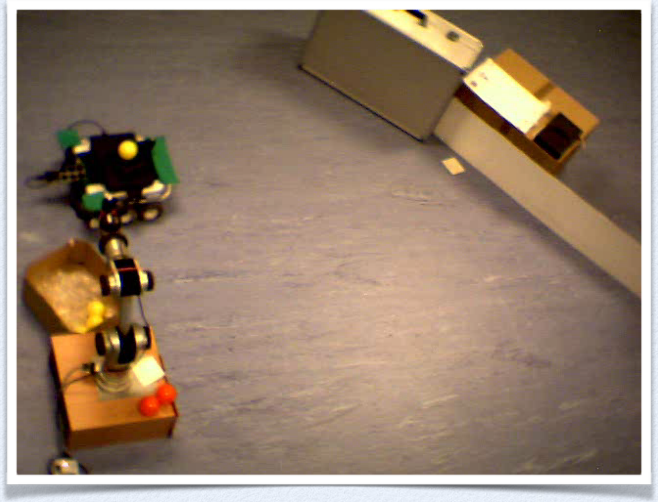
- elaborate and assemble components using **dependencies** (*required services*)

Mapping is a **many to many** relationship, providing **alternatives**

component assembly



adaptation demonstration



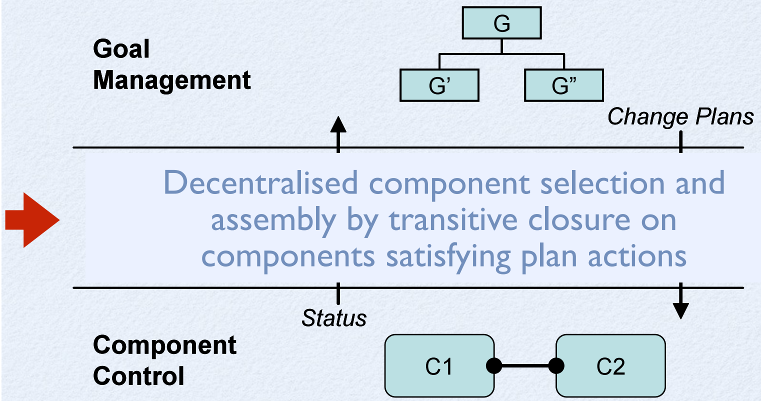
Adaptation may require component reselection or alternative plan selection or replanning

… other assembly adventures …

- **Flashmob** - distributed adaptive self-assembly
  - gossip algorithm
- Exploiting NF preferences in architectural adaptation for self-managed systems
  - component annotations and utility function optimisation

SEAMS 2011, SAC 2010

three layer architecture



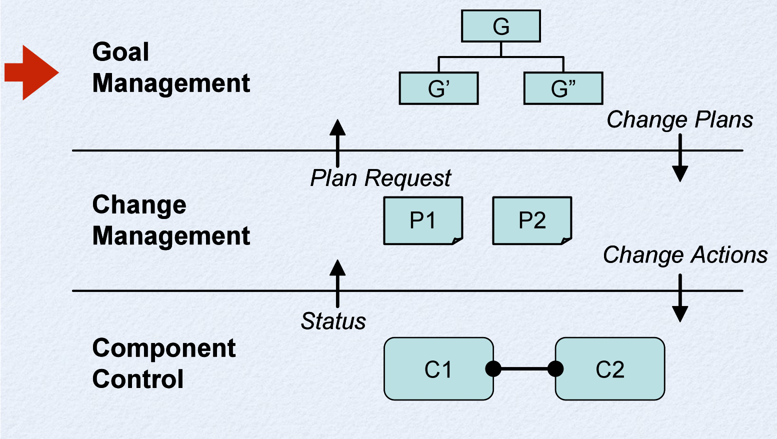
1. Planning over abstract domain

2. Precomputed plans: component assembly and plan execution

3. Component execution and dynamic configuration

ICSE FOSE '07, SEAMS 2008, SEAMS 2011

three layer architecture

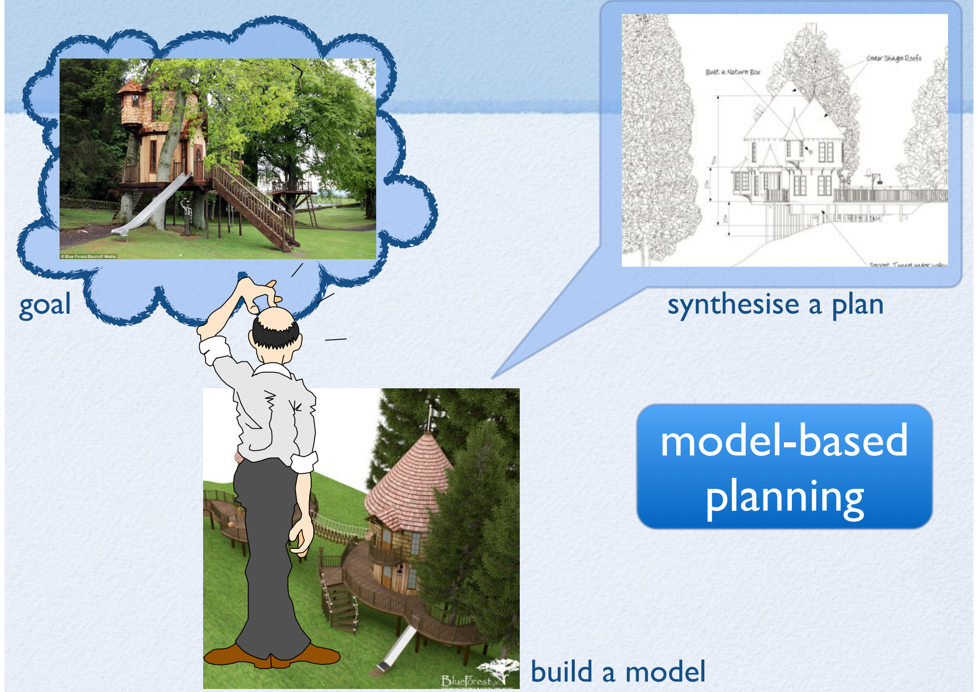


1. Planning over abstract domain

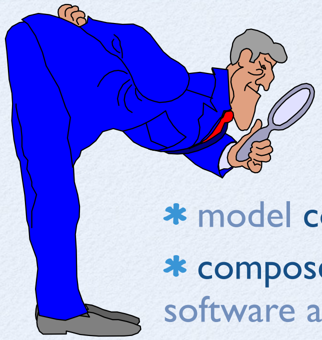
2. Precomputed plans: component assembly and plan execution

3. Component execution and dynamic configuration

ICSE FOSE '07, SEAMS 2008, SEAMS 2011



…earlier modelling adventures…



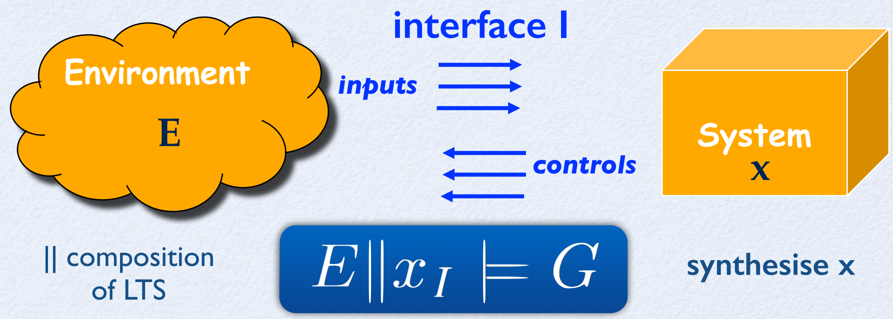
- model component behaviour as **LTS** in **FSP**
- compose behaviours according to the software architecture configuration

… 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.

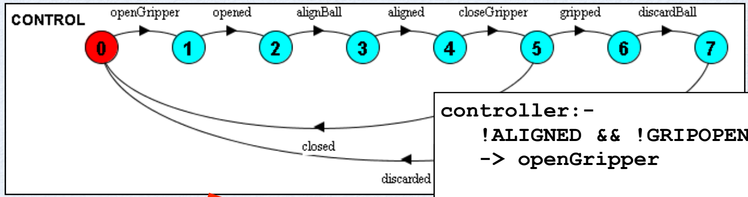


Goal G: Linear Temporal Logic property

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
```

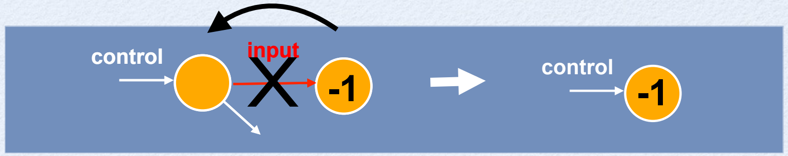
```
ltl_property SAFE4 =
        [](closeGripper -> ALIGNE
ltl_property GETBALL =
        [](alignBall -> X closeGr
ltl_property PROGRESS =
        [](openGripper -> X alignBall)
```

Plan (as a controller)

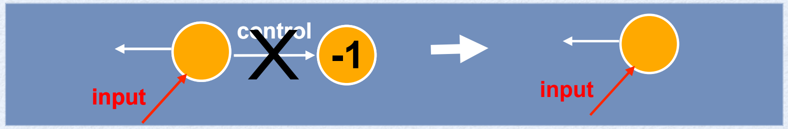
Goal specification (as LTL properties)

computing "winning" states

- By backward propagation of error state for **inputs**:



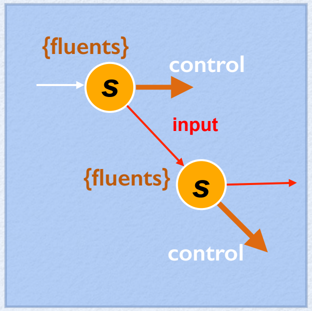
- … for **controls**:



plan extraction

Reactive Plan computed from set of control states **S**
(has outgoing transition labelled with control)

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



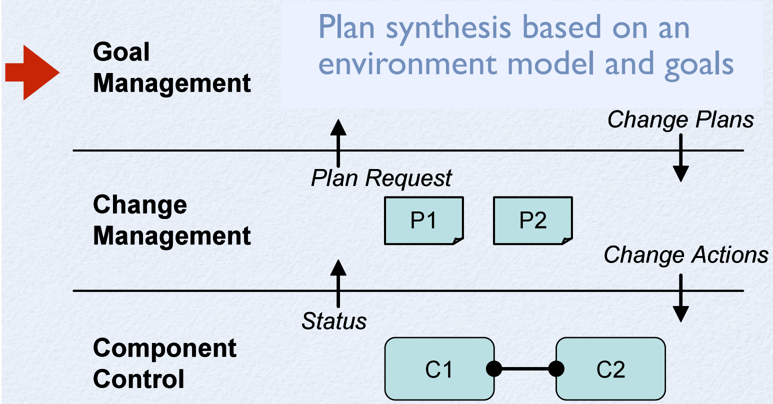

```
controller:-
   !ALIGNED && !GRIPOPEN && !PICKEDUP
   -> openGripper

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

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

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

three layer architecture



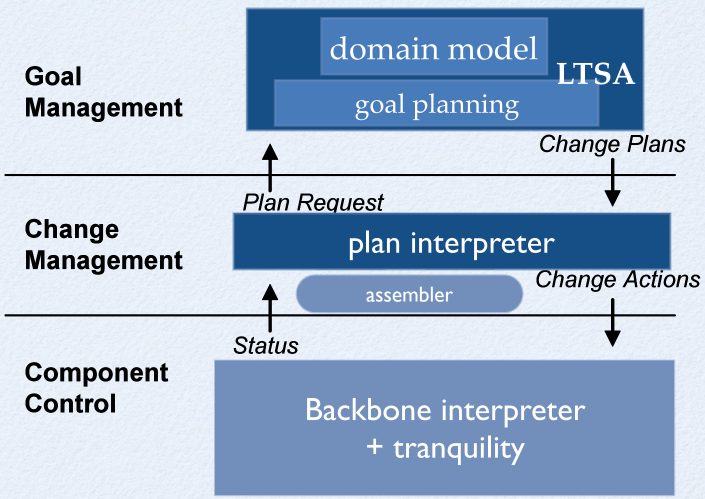
1. Planning over abstract domain

2. Precomputed plans: component assembly and plan execution

3. Component execution and dynamic configuration

ICSE FOSE '07, SEAMS 2008, SEAMS 2011

three layer architecture **realisation**



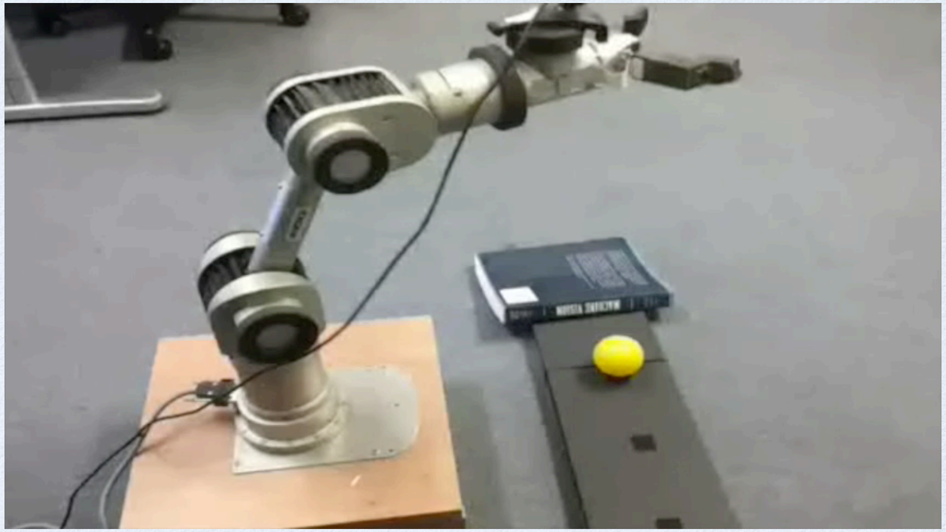
1. Planning over abstract domain

2. Precomputed plans: component assembly and plan execution

3. Component execution and dynamic configuration

ICSE FOSE '07, SEAMS 2008, SEAMS 2011

three layer architecture **realisation**



ICSE FOSE '07, SEAMS 2008, SEAMS 2011

... mostly ...

ICSE 2013 teaser demo

- provided basis for further research …

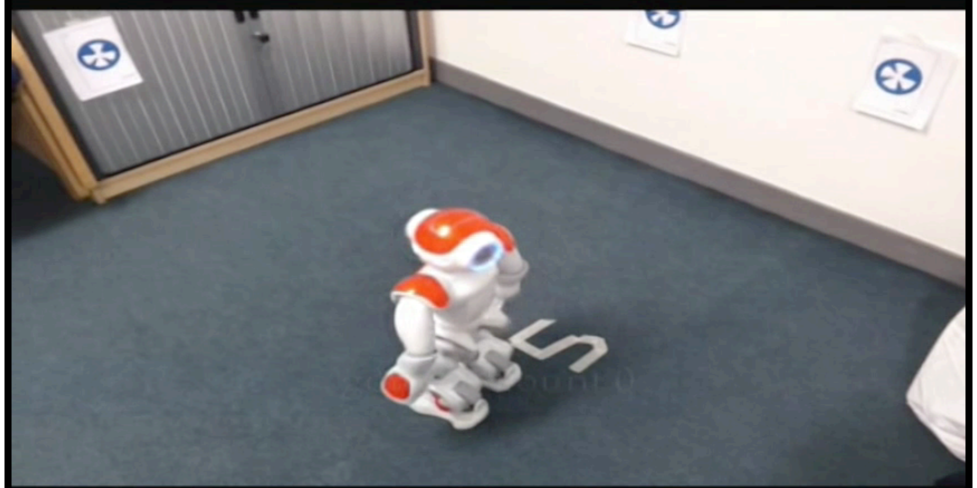
Multi-tier adaptation

idealised	$E_n \| x_{I_n} \models G_n$	strong assumptions and guarantees
	$E_j \| x_{I_j} \models G_j$	
realistic	$E_0 \| x_{I_0} \models G_0$	weak assumptions and guarantees

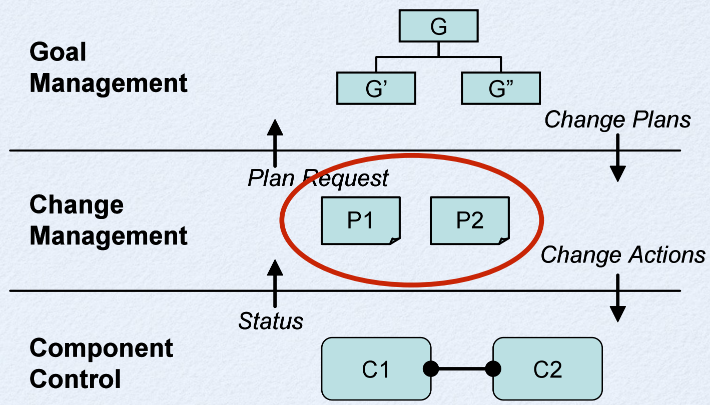
Enhanced Service

Degraded Service

ICSE, 2014 : Hope for the best, plan for the worst…



three layer architecture



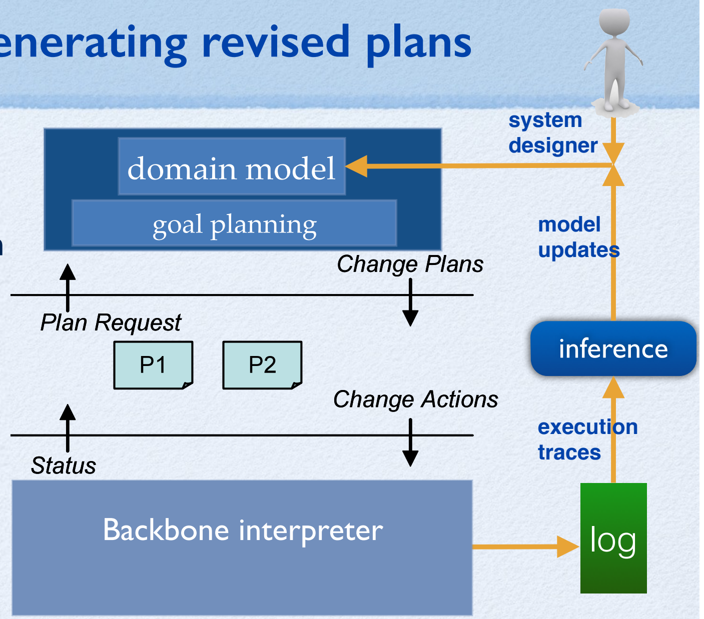
1. *Planning over abstract domain*
2. *Precomputed plans: component assembly and plan execution*
3. *Component execution and dynamic configuration*

ICSE FOSE '07, SEAMS 2008, SEAMS 2011

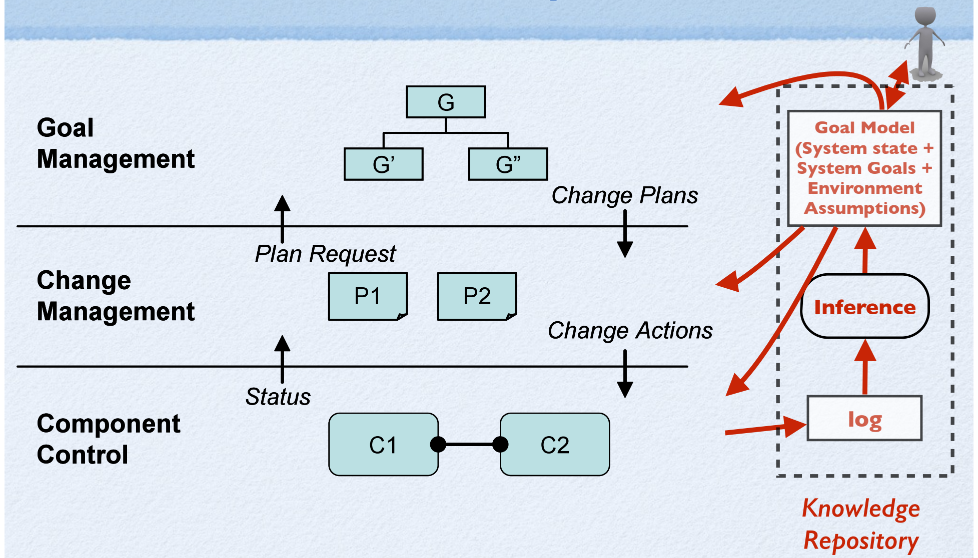
generating revised plans

Plan revision through **domain model revision** *using observations and probabilistic rule learning*

Learning through experience!



elaborate the three layer architecture

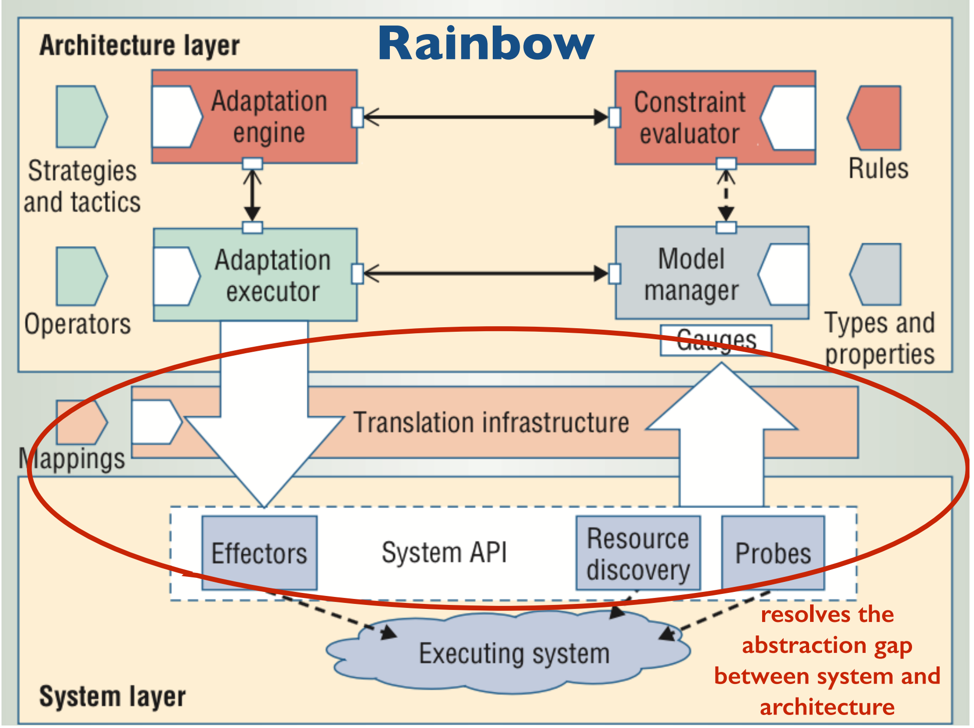


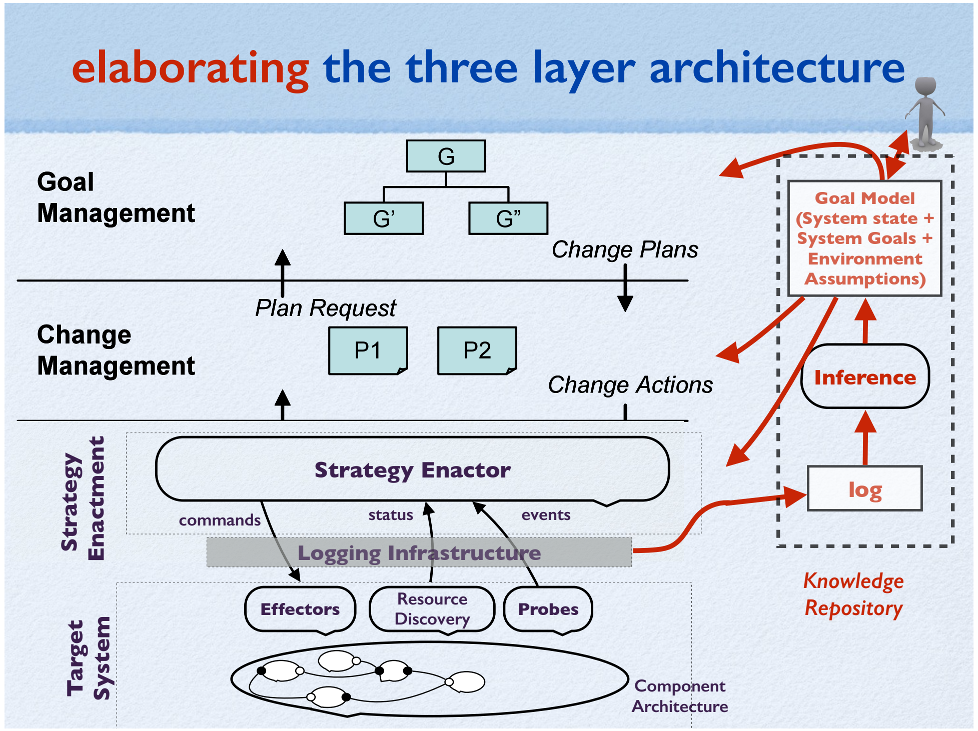
our current vision

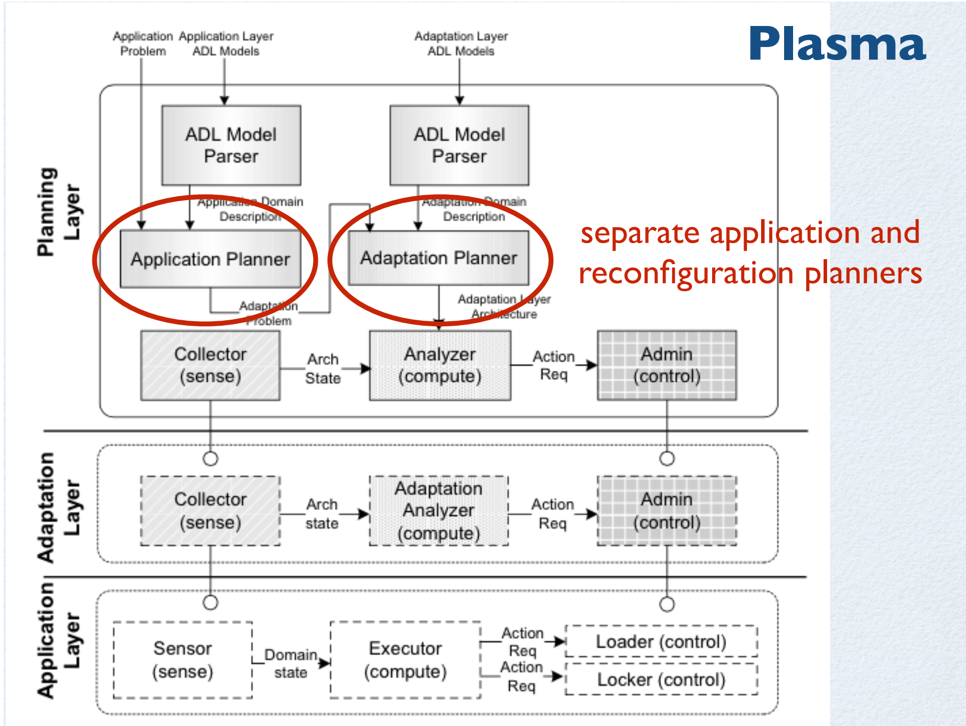
Provide a reference architecture which …

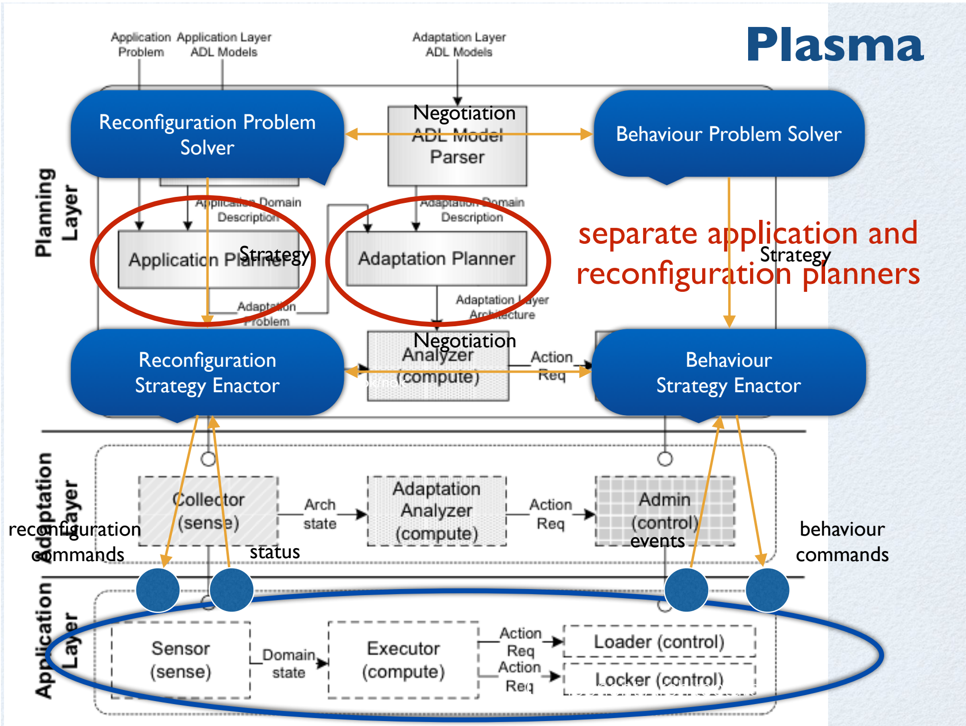
- accommodates specific research aspects more clearly
- facilitates comparison of specific approaches
- provides a pick-and-mix (plug-and-play) architecture

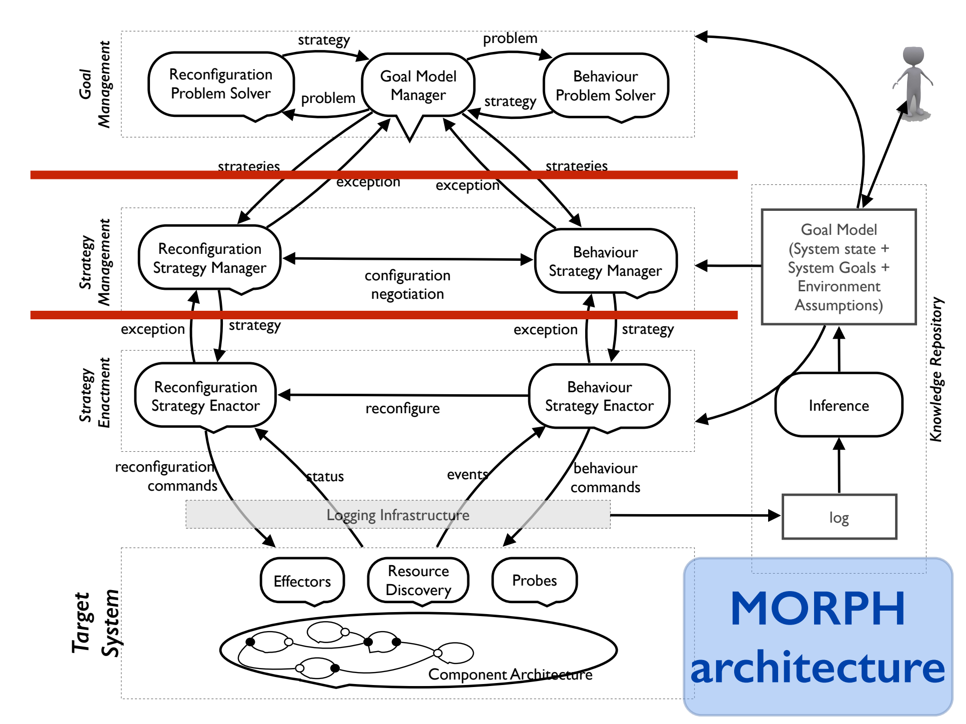
… an adventure playground for software engineers!




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


Plasma
Application Problem
Application Layer ADL Models
Adaptation Layer ADL Models
Planning Layer
ADL Model Parser
ADL Model Parser
Application Domain Description
Adaptation Domain Description
Application Planner
Adaptation Planner
Adaptation Problem
Adaptation Layer Architecture
separate application and reconfiguration planners
Collector (sense)
Arch State
Analyzer (compute)
Action Req
Admin (control)
Adaptation Layer
Collector (sense)
Arch state
Adaptation Analyzer (compute)
Action Req
Admin (control)
Application Layer
Sensor (sense)
Domain state
Executor (compute)
Action Req
Action Req
Loader (control)
Locker (control)


Plasma
Application Problem
Application Layer ADL Models
Adaptation Layer ADL Models
Reconfiguration Problem Solver
Negotiation
Behaviour Problem Solver
Planning Layer
Strategy
Application Planner
Adaptation Planner
separate application and reconfiguration planners
Reconfiguration Strategy Enactor
Negotiation
Behaviour Strategy Enactor
Action Req
Adaptation Layer
Collector (sense)
Arch state
Adaptation Analyzer (compute)
Action Req
Admin (control)
reconfiguration commands
status
events
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

in conclusion ...

Adaptive and Self-Managed Systems

…. the challenge of **change** …

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

the challenge of change

- **model revision** in response to updates and change in the environment
- **online Requirements Engineering** in response to updates and changes in goals (RE@runtime)

■ automated support for diagnosis and repair using a combination of model checking and machine learning

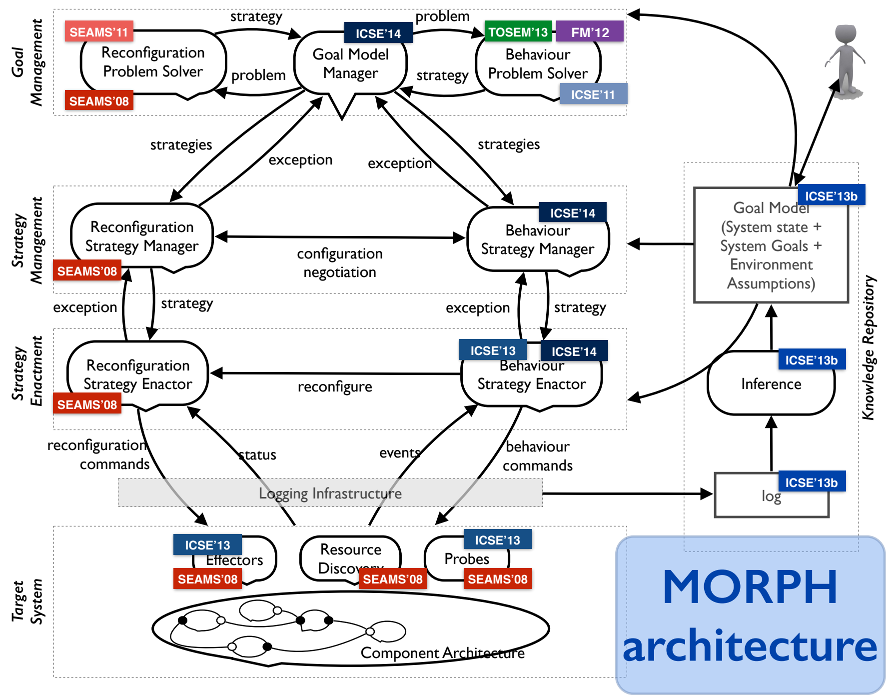
■ automated support for requirements elaboration and obstacle analysis

ASE 2008, ICSE 2009, ICSE 2012, CACM 2015

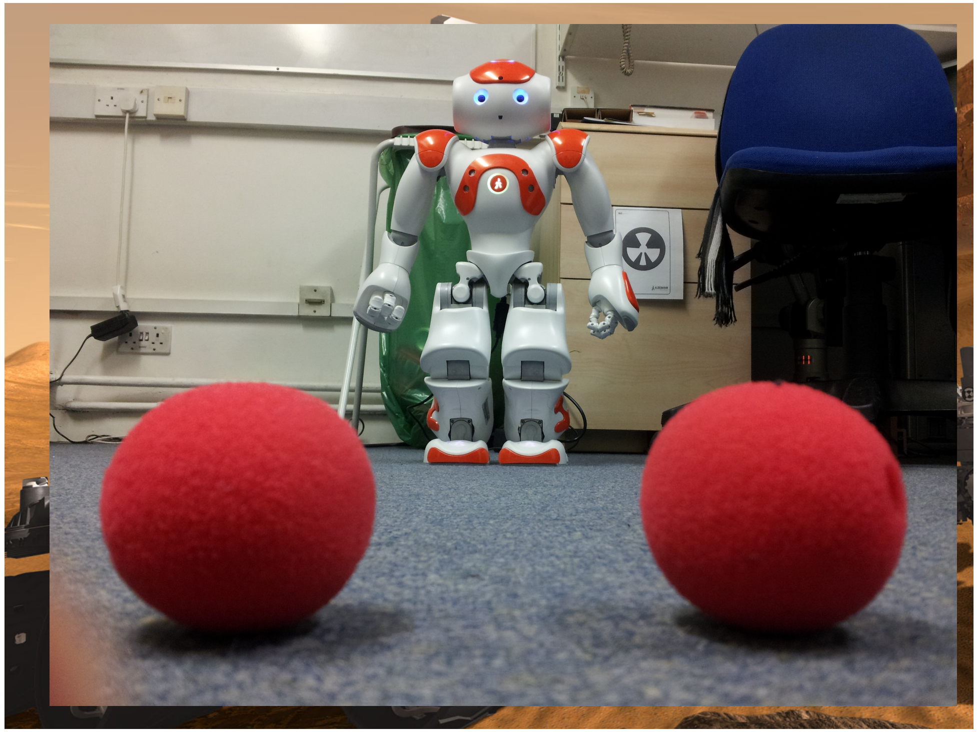
Vision: architectural reference model

- identify and accommodate specific research concerns,
- facilitate comparisons between approaches, and
- provide a framework for potential implementations (plug-and-play)

… an adventure playground for software engineers!


Goal Management
Strategy Management
Strategy Enactment
Target System
SEAMS'11
Reconfiguration Problem Solver
SEAMS'08
strategy
problem
ICSE'14
Goal Model Manager
problem
strategy
TOSEM'13
FM'12
Behaviour Problem Solver
ICSE'11
strategies
exception
exception
strategies
Reconfiguration Strategy Manager
SEAMS'08
configuration negotiation
ICSE'14
Behaviour Strategy Manager
exception
strategy
exception
strategy
Reconfiguration Strategy Enactor
SEAMS'08
reconfigure
ICSE'13
ICSE'14
Behaviour Strategy Enactor
reconfiguration commands
status
events
behaviour commands
Logging Infrastructure
ICSE'13
Effectors
SEAMS'08
Resource Discovery
SEAMS'08
ICSE'13
Probes
SEAMS'08
Component Architecture
ICSE'13b
Goal Model (System state + System Goals + Environment Assumptions)
ICSE'13b
Inference
ICSE'13b
log
Knowledge Repository
MORPH architecture


challenging case studies
evaluation
validation
comparison




Magee
Alessandra Russ
Axel van
msweerde
Katsumi Inoue
Nicholas D'Ippolito
Dalal Alrajeh
Dominico C
ew McVeigh

international cooperation and …

SEAMS

a software engineering adventure playground!

Bliss