

Imperial College
London

Towards Robust Self-Managed Adaptive Systems*

Jeff Kramer

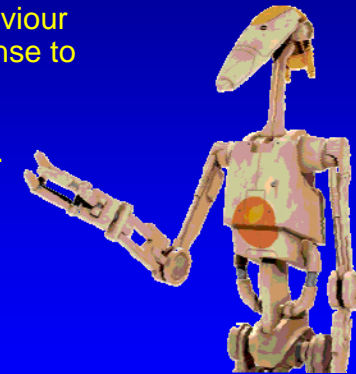
*Based on FOSE'07 paper, Jeff Kramer & Jeff Magee

Current work with Daniel Sykes and Will Heaven

NII Tokyo, April 2008

Self-Managed Systems

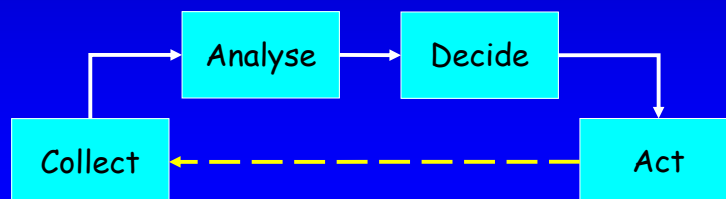
- Autonomous Adaptation
 - Change/update behaviour dynamically in response to changes in goals & environment without operator intervention.
- Self
 - - Configuring
 - - Healing
 - - Tuning



2

Background

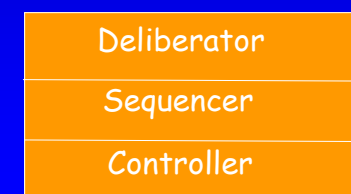
- Conferences
 - Autonomic Computing, SOAS, ICAS
 - CDS, WOSS, SEAMS
- Common Paradigm - single control loop



3

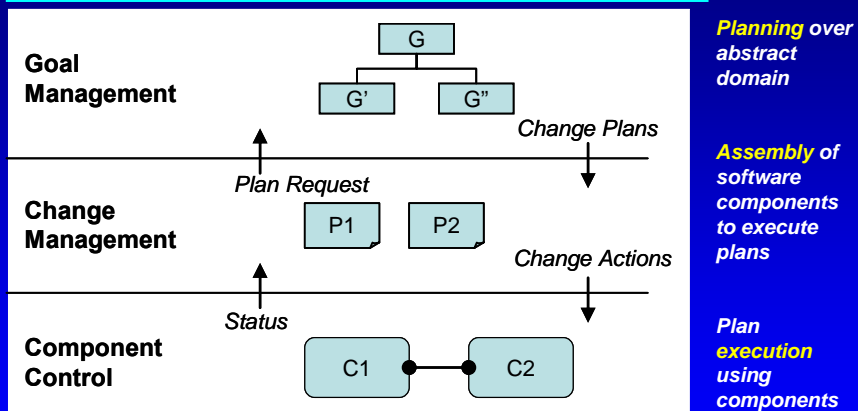
S/W Architecture in Robotics

- SPA 1970's
-
- Three-Layer Architecture (Gat 98)



4

A Three-Layer Architecture Model



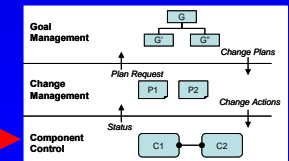
- separation of concerns
- layering according to required response times

5

1. Component Control

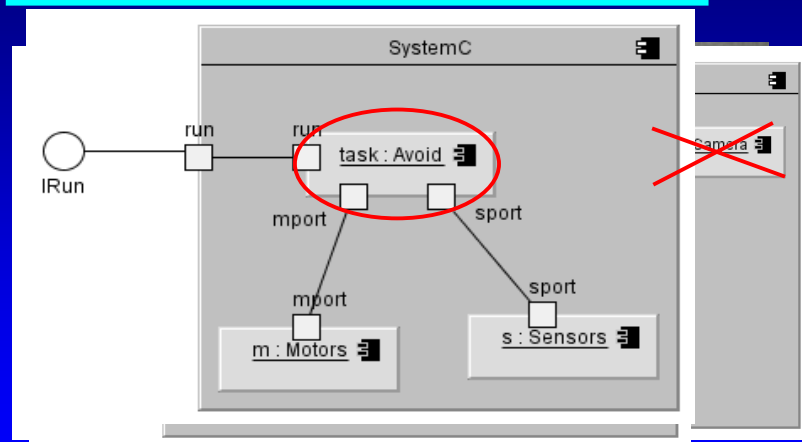
Layer supports

- Dynamic configuration
 - component creation, deletion and connection
 - Event/status reporting during change
 - Probes & Effectors
- Component execution
- Component self-tuning
 - e.g. TCP timeouts, collision avoidance



6

Component Control - implementation



7

Component Control - Research Challenges

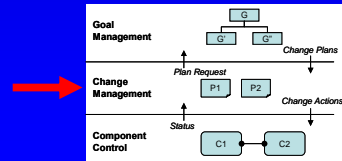
- Safe operation during change
 - stable conditions (quiescence) – Kramer & Magee 1986
 - avoid control transients – Schaefer & Wehrheim 2007
- Verification of safety properties during change
 - Zhang & Cheng 2006

8

2. Change Management

Layer supports

- Plan execution
 - in response to predicted class of events/ state changes in the underlying layer e.g. component failure, mode change.
- Component selection and configuration management
- Plan update
 - in response to unpredicted change (eg. goals)



9

Plans - implementation

- **Reactive Plans** are described in terms of **condition-action rules** over an alphabet of **plan actions**

```
at(loc1) & !loaded  
-> pickup
```

```
at(loc1) & loaded  
-> moveto(loc2)
```

```
at(loc2) & loaded  
-> putdown
```

```
at(loc2) & !loaded  
-> moveto(loc1)
```

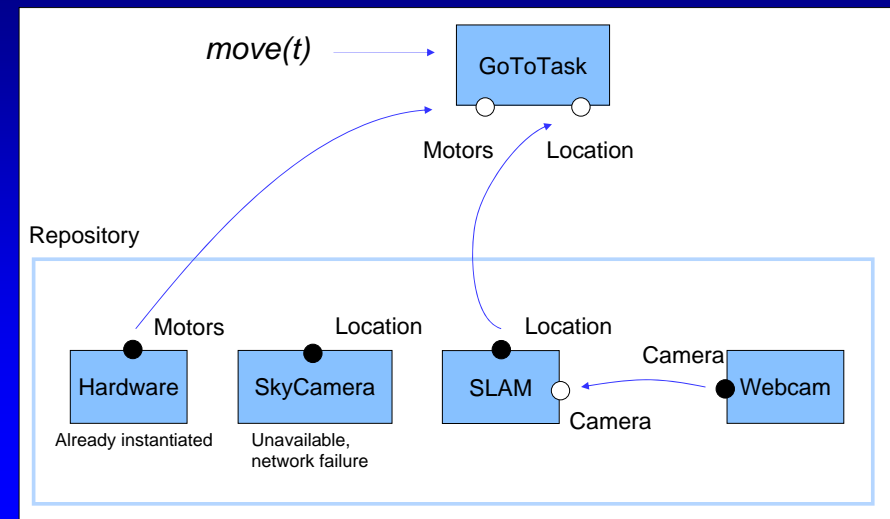
Includes **alternative** paths to the goals should the environment change in an unpredictable manner.

10

Deriving configurations

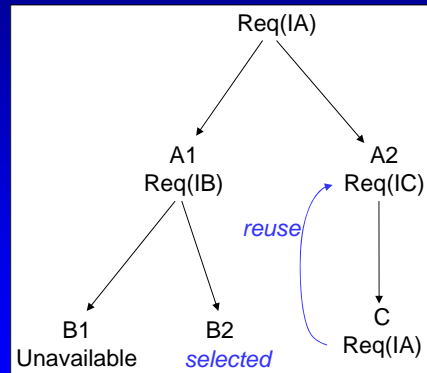
- **Plan actions** do not refer to configurations explicitly
- Primitive **actions** associated with **interfaces** which the interpreter can call
- Hence, need a set of **components** which implement every **interface** required by the plan, elaborated using **dependencies**
- Components to interfaces is a **many to many** relationship, providing alternatives

Component selection



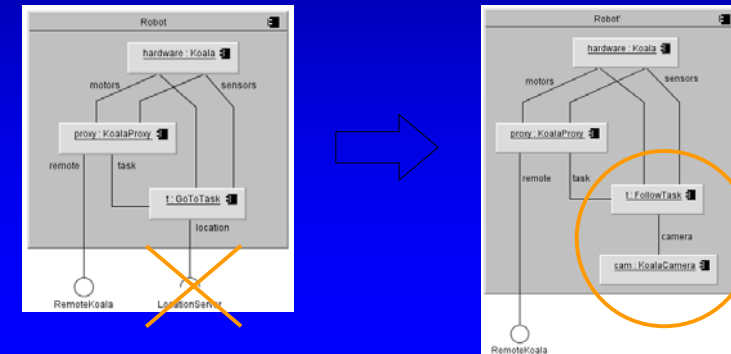
Component selection

- Components already instantiated or already selected are reused
 - Assumes one instance providing each interface
- Components marked as unavailable (or have unsatisfiable requirements) are not selected
- Here, 2 solutions – {A1,B2} or {A2,C} – which is better?
 - Use NF attributes to choose



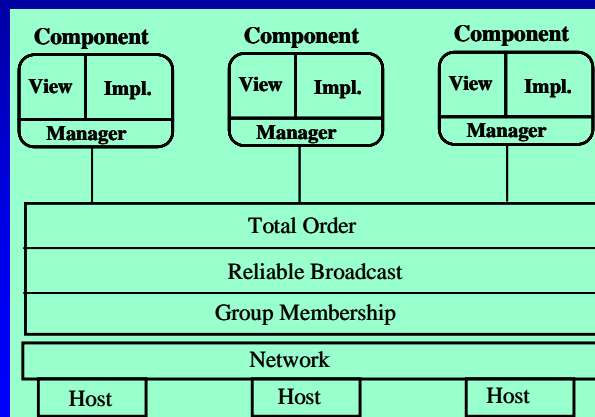
Adaptation

- Components that 'fail' at runtime invoke the **selection** process
- 'Failed' component marked as unavailable
- If no alternatives can be used, **replanning** may be necessary



Change Management – Research Challenges

- Scalability -> Distribution & Decentralisation



Georgiadis
2002 -
Decentralized
but not
Scalable

3. Goal Management

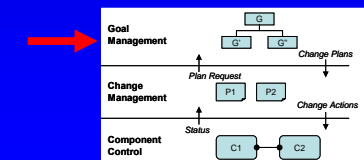
- Layer supports **plan generation** in response to
 - addition/removal of goals
 - requests from below, due to plan failure

System \neq Goals

Planner (System, Goals) \rightarrow Plan

Plan (System) \rightarrow System'

System' \neq Goals



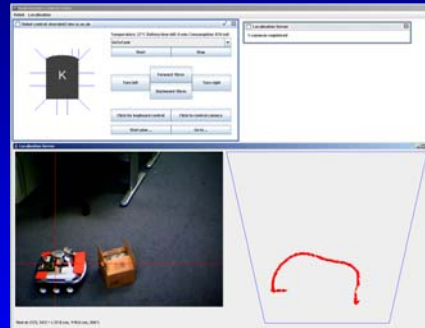
Goals - example

Goals

- Go to X pickup Y
- Follow X
- Rescue Y

Constraints

- Rules of engagement
- Resources (e.g. using two robots)
- Capabilities (sensors, actuators)



17

System changes - unintentional

Environment

- Obstacles
- context
- ...

Failures

- Sensors
- Actuators
- Software



rather than avoiding obstacles, Rupert decided the optimal course of action would be to drive through them

18

Planning - implementation

Domain description describes world and system capabilities.
Goal given in temporal logic

```
pre: !at(loc)
action: moveto(loc)
post: at(loc)
```

```
pre: !loaded &
at(loc1)
action: pickup
post: loaded
```

```
pre: loaded & at(loc2)
action: putdown
post: !loaded
```

Planning as model checking

Replanning

Reactive plan generated by backtracking from goal states



```
at(loc1) & !loaded
-> pickup
```

```
at(loc1) & loaded
-> moveto(loc2)
```

```
at(loc2) & loaded
-> putdown
```

```
at(loc2) & !loaded
-> moveto(loc1)
```

19

Goal Management – Research Challenges

Precise specification of domain model and goals

- application goals
- system goals
- covering structure, behaviour, performance ...

Goal refinement

Runtime Goal & Constraint Checking

Planning

- Scalability → Hierarchical Decomposition

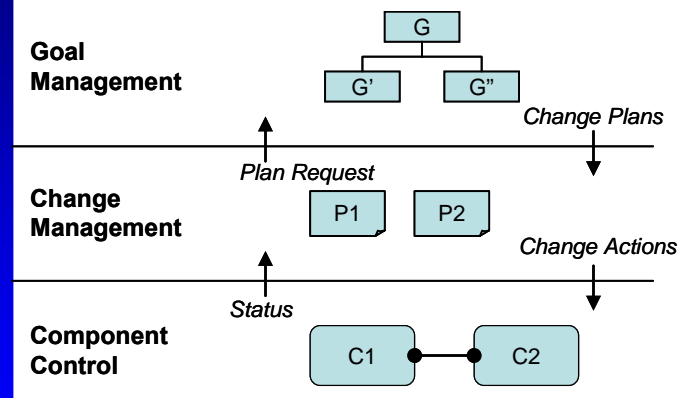
20

Example Scenario



21

A Three-Layer Architecture Model



Planning over abstract domain

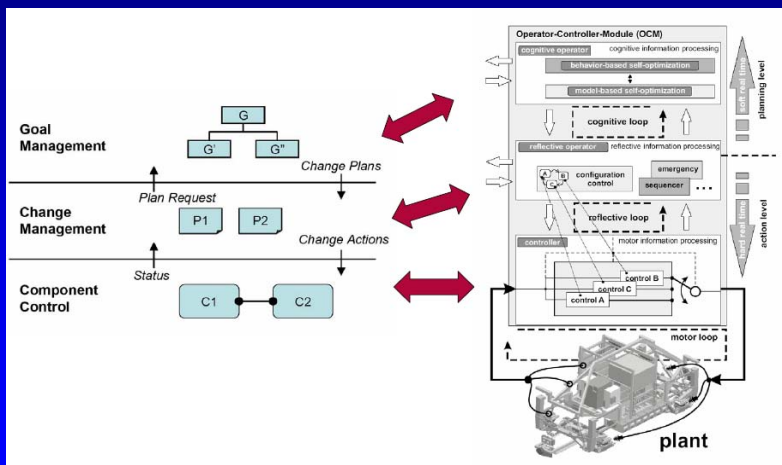
Assembly of software components to execute plan

Plan execution using components

- separation of concerns
- layering according to required response times

22

Reference Model - implementations

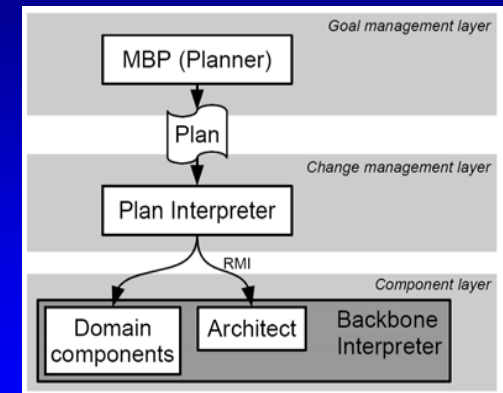


Self-optimising Mechatronic System (Railcab) – Holger Giese

23

Implementation - Status

- Plan interpreter
 - Currently runs on a desktop machine
- Component selection
 - Selection not yet fully integrated with plan interpreter
- Components
 - implemented in Java, running on top of the Backbone system, directly on the Koala robots



Overall SE Research Challenges

- Challenge is to automate and run on-line what are currently off-line RE/design processes e.g. goal-refinement...
 - Planning as model-checking
- Need to decide for a given application the requirement for adaptability etc. and the level of automation needed.
- Need to cope with incomplete information about the environment.
(not incomplete goals?)

25

Questions



26