...the challenge of change...

evironment $E$
goals $G$
capabilities $I$

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

Off-line ...
requirements analysis,
design, implementation,
redeployment ....

Run-time ...
adaptation to unforeseen changes ....

Adaptive and Self-Managed Systems

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

$E || x_I || G$ $E' || x'_I || G'$

...to automate and run on-line what is currently done off-line!
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 unforeseen changes in the environment, goals and/or capabilities of the system

Disruptive change!
architecture is important

three layer architecture

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

three layer architecture

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

MAPE cycle

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

Monitor - Analyse - Plan - Execute

sensors - effectors
inspiration from robotics

1970's

Sense Plan Act

Deliberator Sequencer Controller

layering according to response times

1998 (Gat)

1. Planning
2. plan execution
3. component feedback control

layers according to response times

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

… some of our earlier research …

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

Compile, build and deploy

Compile, build and deploy

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 (i.e. the environment is passive)

behaviour consistency


TSE 1985

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
**Plan Execution**

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

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

**Reactive Plans**

- **Condition-Action Rules**
  - Over an alphabet of plan actions

**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
Adaptation may require component reselection or alternative plan selection or replanning.

- **Flashmob** - distributed adaptive self-assembly
  - gossip algorithm

- Exploiting NF preferences in architectural adaptation for self-managed systems
  - component annotations and utility function optimisation

**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
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

\begin{align*}
\text{ltl_property SAFE} &= [\square (\text{closeGripper} \implies \text{ALIGNED})] \\
\text{ltl_property GETBALL} &= [\square (\text{alignBall} \implies \text{X closeGripper})] \\
\text{ltl_property PROGRESS} &= [\square (\text{openGripper} \implies \text{X alignBall})]
\end{align*}

**Plan (as a controller)**

- controller: $\neg \text{ALIGNED} \land \neg \text{GRIPOPEN} \land \neg \text{PICKEDUP}$ $\implies$ openGripper
- $\neg \text{ALIGNED} \land \text{GRIPOPEN} \land \neg \text{PICKEDUP}$ $\implies$ alignBall
- $\neg \text{ALIGNED} \land \neg \text{GRIPOPEN} \land \text{PICKEDUP}$ $\implies$ discardBall
- $\text{ALIGNED} \land \text{GRIPOPEN} \land \neg \text{PICKEDUP}$ $\implies$ closeGripper

**Environment model (as $\parallel$ LTS)**

**Plan (controller) synthesis**

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

...earlier modelling research...

**model-based planning**

build a model

synthesize a plan

...model check properties using LTSA


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.

three layer architecture

three layer architecture realisation

Goal Management

Plan synthesis based on an environment model and goals

1. Planning over abstract domain

Change Management

Plan Request

P1

P2

2. Precomputed plans: component assembly and plan execution

Component Control

Status

C1

C2

3. Component execution and dynamic configuration

Goal Management

Change Plans

domain model

LTSA

1. Planning over abstract domain

Change Management

Plan Request

Plan interpreter

Assembler

Backbone interpreter + tranquility

2. Precomputed plans: component assembly and plan execution

Component Control

Status

C1

C2

3. Component execution and dynamic configuration
three layer architecture realisation

ICSE FOSE ’07, SEAMS 2008, SEAMS 2011

ICSE 2013 teaser demo

Success.

... mostly ...

provided basis for further research ...
Multi-tier adaptation

Idealised

\[ F_n \parallel x_n \Rightarrow G_n \]

Strong assumptions and guarantees

Realistic

\[ F_0 \parallel x_0 \Rightarrow G_0 \]

Weak assumptions and guarantees

Enhanced Service

Degraded Service

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

Generating revised plans

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

Learning through experience!

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

Domain model

Goal planning

Inference

Backbone interpreter

ICSE 2013

ICSE FOSE '07, SEAMS 2008, SEAMS 2011
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

… a playground for adaptive engineers!

elaborating the three layer architecture

Rainbow

resolves the abstraction gap between system and architecture

Plasma

separate planners for application behaviour and reconfiguration
Plasma

separate planners for application behaviour and reconfiguration

Vision: architectural reference model

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

- evaluation
- validation
- comparison

Requirements Engineering

World

- Environment assumptions
- Requirements

Interface

- Specification

Machine

- E, x_i ⊨ R

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

Adaptive and Self-Managed Systems

.... the challenges of change ...

environment
goals
capabilities

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

.... a sound foundation can be provided by an appropriate architecture.

in conclusion ...

AWASE

architecture provides an adaptive engineering playground!