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 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

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

- Analyse
- Plan
- Execute
- Monitor

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

sensors

effectors

Monitor

Analyse

Plan

Execute

inspiration from robotics

- 1970’s
- 1998 (Gat)
  1. Planning
  2. Plan execution
  3. Component feedback control

layering according to response times

Sense

Plan

Act

Deliberator

Sequencer

Controller

three layer architecture

TD: decreasing statefullness and strategic planning

Goal Management

G

G'

G''

Change Plans

Change Management

Plan Request

P1

P2

Change Actions

Component Control

Status

C1

C2

BU: increasing response time

a separation of concerns

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

1970’s inspiration from robotics

three layer architecture

a separation of concerns

ICSE FOSE ’07
CONIC and Darwin

- distributable, context-independent components
- interaction via a well-defined interface
- an explicit configuration description (ADL)
- third party instantiation and binding

How can we do this safely?

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

on-line dynamic change

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

configuration consistency

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


TSE 1985
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. environment is passive)

General change model:

**Passive**
- create
- bind
- activate
- passivate

**Active**
- delete
- unbind

**Component States**

Three layer architecture:

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

Safe operation, including during change (quiescence)
Reactive plans

- condition-action rules over an alphabet of plan actions

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

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

three layer architecture

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

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

...earlier modelling adventures...

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

... model check properties using LTSA

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

plan (controller) synthesis

Environment model (as || LTS)

controller :-

!ALIGNED && !GRIPOPEN && !PICKEDUP
-> openGripper

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

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

ALIGNED && GRIPOPEN && !PICKEDUP
-> closeGripper

Input

Plan (as a controller)

Goal specification (as LTL properties)

ltl_property SAFE4 = (closeGripper -> ALIGNED)

ltl_property GETBALL = (alignBall -> X closeGripper)

ltl_property PROGRESS = (openGripper -> X alignBall)

three layer architecture

Plan synthesis based on an environment model and goals

1. Planning over abstract domain

2. Precomputed plans: component assembly and plan execution

3. Component execution and dynamic configuration

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.

116x317 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.

116x317 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.

116x317 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 realisation

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

Success.

... mostly ...
Multi-tier adaptation

idealised
\[ E_n \left| x_I_n \right| = G_n \]
strong assumptions and guarantees

realistic
\[ E_0 \left| x_I_0 \right| = G_0 \]
weak assumptions and guarantees

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

! provided basis for further research …
generating revised plans

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

Learning through experience!

domain model

goal planning

Backbone interpreter

system designer

model updates

inference

executes traces

Log

elaborate the three layer architecture

Goal Model (System state + System Goals + Environment Assumptions)

Goal Management

Change Management

Component Control

Knowledge Repository

Inference

log

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!

Rainbow

resolves the abstraction gap between system and 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

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!

challenging case studies

- evaluation
- validation
- comparison

MORPH architecture
international cooperation and ...

competition!

acknowledgement

SEAMS

a software engineering adventure playground!

Bliss