Jeff Kramer
Imperial College London

SEFM 2017

... the challenge of change ...

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 Adaptive Systems

Disruptive change!

Self-Managed Adaptive Systems

... being rigorous is essential!

... more formally ...

\[ E \parallel x_I \models G \]

\[ E' \parallel x'_I \models G' \]

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

change adapt change change
Self-Managed Adaptive Systems

models @ runtime

... in an appropriate architecture with a rich runtime environment

three layer architecture

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

architecture is important

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

CONIC and Darwin

- on-line dynamic change

Once installed, the software could be dynamically modified without stopping the entire system.

Composite Component

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

Compiled, build and deploy

structure specification

structural specification

evolved structural specification

change script

preserve consistency

system

evolved system

behaviour consistency

Component States

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.

Active: the component is active and the environment is active i.e. transactions will be initiated on it.

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

ICSE FOSE ’07, SAVCBS 2007, SEAMS 2008
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 unpredictable 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

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.

Reactive plans with component selection and assembly by transitive closure on components satisfying plan actions.

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

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

Model-based planning

Build a model

Synthesise a plan

Three layer architecture

Goal Management

Change Plans

G

G’

G”

Change Actions

Plan Request

P1

P2

Component Control

Status

C1

C2

Goal Management

Synthesise a plan

Model-based planning

Build a model

Three layer architecture

Goal Management

Change Plans

G

G’

G”

Change Actions

Plan Request

P1

P2

Component Control

Status

C1

C2

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

- 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.

Environment model (as $E \parallel \text{LTS}$)

Goal specification (as LTL properties)

Plan (as a controller)


Environment $E$

inputs

composition of LTS $E \parallel x I = G$

controls

synthesize $x$

Goal G: Linear Temporal Logic property

Controller synthesis

ltl_property SAFE4 = $\left[ \right] \left( \text{closeGripper} \implies \text{ALIGNED} \right)$

ltl_property GETBALL = $\left[ \right] \left( \text{alignBall} \implies \text{closedGripper} \right)$

ltl_property PROGRESS = $\left[ \right] \left( \text{openGripper} \implies \text{X alignedBall} \right)$

controller:

- $\text{opened} \land \text{GRIPOPEN} \land \text{PICKEDUP} \implies \text{openGripper}$
- $\text{ALIGNED} \land \text{GRIPOPEN} \land \text{PICKEDUP} \implies \text{alignBall}$
- $\text{ALIGNED} \land \text{!GRIPOPEN} \land \text{PICKEDUP} \implies \text{discardBall}$
- $\text{ALIGNED} \land \text{GRIPOPEN} \land \text{!PICKEDUP} \implies \text{closeGripper}$
computing “winning” states

- By backward propagation of the error state \(-1\) for inputs from the environment:

![Diagram showing backward propagation of error state from input to control]

- By removal of the error state \(-1\) for controls from the controller:

![Diagram showing removal of error state from control]

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: -

\[
\begin{align*}
&!ALIGNED \& !GRIPOPEN \& !PICKEDUP \rightarrow \text{openGripper} \\
&!ALIGNED \& !GRIPOPEN \& !PICKEDUP \rightarrow \text{alignBall} \\
&!ALIGNED \& !GRIPOPEN \& !PICKEDUP \rightarrow \text{discardBall} \\
&ALIGNED \& GRIPOPEN \& !PICKEDUP \rightarrow \text{closeGripper}
\end{align*}
\]

three layer architecture realisation

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

domain model

LTSA

ICSE FOSE ’07, SEAMS 2008, SEAMS 2011
three layer architecture \textit{realisation}

ICSE FOSE ’07, SEAMS 2008, SEAMS 2011

Success.

… mostly …

shortcomings provide the challenges for further research …

ICSE 2013 teaser demo
Multi-tier adaptation

idealised  \[ E_n \| x_n \|_n \Rightarrow G_n \]

strong assumptions and guarantees

realistic  \[ E_0 \| x_0 \|_0 \Rightarrow G_0 \]

weak assumptions and guarantees

Enhanced Service

Degraded Service

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

Planning over abstract domain

Precomputed plans: component assembly and plan execution

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!

Backbone interpreter

elaborating the three layer architecture
Rainbow resolves the abstraction gap between system and architecture.

Plasma elaborates the three layer architecture, with separate planners for application behaviour and reconfiguration.

MORPH architecture
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

our architectural vision
In general we need to tailor the transition properties $T$. 

- Hot Swap
- Stop Old Spec
- Start New Spec

Dynamic Controller Update
In general we need to tailor the transition properties. The Dynamic Controller Update prescribes:

- StopOldSpec
- StartNewSpec & Reconfigure

**Dynamic Controller Specification**

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

**Dynamic Controller Transition**

Transition properties must be elicited:

1. \( G \models \text{stopOldSpec} \)
2. \( T \)
3. \( \Box (\text{startNewSpec} \implies \Box G') \)
4. \( \Box (\text{hotSwap} \implies (\Diamond \text{stopOldSpec} \land \Diamond \text{reconfigure} \land \Diamond \text{startNewSpec})) \)
**Dynamic Controller Hotswap**

\[
E \mid x_I \models G
\]

\[
E' \mid x'_I \models G'
\]

**Enactor**

\[
x_I
\]

\[
x'_I
\]

\[
T
\]

**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

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

... need to use rigorous techniques and 
formal methods

SEFM

... an appropriate architecture

a sound foundation and context for research.

Bliss