Chapter 8. Dynamic and Adaptive Systems

**Structural View - Darwin ADL**

- Distributable context independent components with interaction via well defined interfaces
- An explicit configuration description (ADL) in terms of instantiation and binding

**Online dynamic change**: Once installed, the software could be dynamically changed 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? Can we maintain application configuration consistency and behaviour consistency during change?
**Configuration Consistency**

- Construction/implementation
- system
- evolved structural description
- change script
- evolved system

*e.g. Conic, Regis*

---

**Behaviour Consistency**

**Component States**

- Passive
- Active

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

**A Passive component**
- is consistent with its environment, and
- services interactions, but does not initiate them.

---

**Change Rules for behaviour consistency**

**Quiescent** – passive and no transactions are in progress or will be initiated.

- **Operation**     | **Pre-condition**
  - delete          | component is quiescent and isolated
  - bind/unbind     | connected component is quiescent
  - create          | true

---

**Example - a simplified RING Database**

Nodes perform autonomous updates

Updates propagate round the ring via channels
RING Required Properties (1)

// node is PASSIVE if passive signalled and not yet changing or deleted
fluent PASSIVE[i:Nodes]
  = <node[i].passive,
      node[i].{change[Value],delete}>

// node is CREATED after create until delete
fluent CREATED[i:Nodes]
  = <node[i].create, node[i].delete>

// system is QUIESCENT if all CREATED nodes are PASSIVE
assert QUIESCENT
  = forall[i:Nodes] (CREATED[i] -> PASSIVE[i])

RING Required Properties (2)

// value for a node i with color c
fluent VALUE[i:Nodes][c:Value]
  = <node[i].change[c], ...>

// state is consistent if all created nodes have the same value
assert CONSISTENT
  = exists[c:Value] forall[i:Nodes]
      (CREATED[i] -> VALUE[i][c])

// safe if the system is consistent when quiescent
assert SAFE = [] (QUIESCENT -> CONSISTENT)

// live if quiescence is always eventually achieved
assert LIVE = []<> QUIESCENT

Current Research ...

Self-managed adaptive change

Self-Managed Adaptive Systems

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

- Self
  - Configuring
  - Healing
  - Tuning

WOSS 2003
Example: robotics

S/W Architecture in Robotics

- SPA 1970’s
  - Sense → Plan → Act
- Three-Layer Architecture (Gat 98)
  - Deliberator
  - Sequencer
  - Controller

A Three-Layer Architecture Model

1. Component Control Layer

Layer supports
- Component execution
- Component self-tuning
  - e.g. TCP timeouts, collision avoidance
- Dynamic configuration
  - component creation, deletion and binding
  - Event/status reporting during change

- separation of concerns
- layering according to required response times

FOSE (ICSE) 2007
2. Change Management Layer

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

Plan execution

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

\[
\text{AT.loc1} \land \neg \text{LOADED} \rightarrow \text{pickup}
\]
\[
\text{AT.loc1} \land \text{LOADED} \rightarrow \text{moveto.loc2}
\]
\[
\text{AT.loc2} \land \neg \text{LOADED} \rightarrow \text{putdown}
\]
\[
\text{AT.loc2} \land \text{LOADED} \rightarrow \text{moveto.loc1}
\]
Component assembly: deriving configurations

- Primitive plan actions (*pickup, moveto, …*) are associated with the provided services at component interfaces which the interpreter can call (*pickup, moveto, …*)
- Elaborate and assemble components using dependencies (required services)
- Hence, need a set of components which implement every interface required by the plan
- Components to interfaces is a many to many relationship, providing alternatives

Component selection

Adaptation Demonstration

Adaptation may require component reselection OR Alternative plan selection OR replanning

Change Management – Research Challenges

- Scalability -> Distribution & Decentralisation
  - Georgiadis 2002 - Imposed total ordering
    Decentralized but not Scalable
  - Daniel Sykes 2010 - Flashmob - gossip algorithm with convergence
  - Daniel Sykes 2010 - component annotations and utility function optimisation for NF preferences
3. Goal Management Layer

- Layer supports plan generation in response to
  - addition/removal of goals
  - requests from below, due to plan failure, often as a result of changes in context (environment)

---

Plan Synthesis

Consider 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 order of inputs from environment.

Environment $E$ and System $x$ with interface $I$

Model as $||$ composition of LTS

Synthesise a reactive plan

$E || x_I \models G$

G is an F LTL property

---

Example

Goal: Controller of the cat and mouse flaps such that ensuring cat and mouse are never in the same room.

---

Environment Representation

Environment: $||$ composition of LTS

MOUSE

0 1 2 3 4 5 6 7 8 9 10 11


m3 mouse_in[4] m5

mouse_in[3] m6

mouse_in[0] m4

mouse_in[2] m2

mouse_in[1] m1

---
Goal Representation

Goal: Linear Temporal Logic property

\[
\text{ltl\_property \-safe} = \neg \exists x \forall y \left( \text{CATROOM}[x] \land \text{MOUSEROOM}[y] \right)
\]

Fluents:

\[
\begin{align*}
\text{fluents CATROOM}[\text{room}:0..4] &= <\text{cat}\_\text{in}[\text{room}], \text{cat}\_\text{in}[0..4] \setminus \{\text{cat}\_\text{in}[\text{room}]\}> \\
\text{fluents MOUSEROOM}[\text{room}:0..4] &= <\text{mouse}\_\text{in}[\text{room}], \text{mouse}\_\text{in}[0..4] \setminus \{\text{mouse}\_\text{in}[\text{room}]\}>
\end{align*}
\]

Plan Synthesis*

\[
Q = \text{set of states} \\
F = \text{set of accepting states (G holds)} \\
F^* = \text{set of winning states found iteratively such that transition out of } F^* \text{ is via a controlled action.}
\]

* Symbolic Controller Synthesis, Asarin, Maier, Pnueli, 1989

Goal Representation - LTS

Safety Property Automata

Computing \(F^*\)

\[
Q = (\text{CAT} \mid \text{MOUSE} \mid \text{SAFE})
\]

\text{Compute } F^* \text{ by backward propagation of error state:}

finally

\[
\text{control} \xrightarrow{-1} \text{control} \xrightarrow{-1}
\]

\[
\text{input} \xrightarrow{-1} \text{input}
\]
Reactive Plan

- **controller**:
  - CATROOM.0 MOUSEROOM.1 -> c4
  - CATROOM.0 MOUSEROOM.2 -> {c1, c4, m2}
  - CATROOM.0 MOUSEROOM.3 -> c1
  - CATROOM.0 MOUSEROOM.4 -> {c1, c4, m5}
  - CATROOM.1 MOUSEROOM.0 -> {c2, c7b, m1, m4}
  - CATROOM.1 MOUSEROOM.2 -> c7b
  - CATROOM.1 MOUSEROOM.3 -> {c2, m6}
  - CATROOM.1 MOUSEROOM.4 -> {c2, c7b, m5}
  - CATROOM.2 MOUSEROOM.0 -> m4
  - CATROOM.2 MOUSEROOM.3 -> {c3, m6}
  - CATROOM.2 MOUSEROOM.4 -> {c3, m5}
  - CATROOM.3 MOUSEROOM.0 -> {c5, c7a, m1, m4}
  - CATROOM.3 MOUSEROOM.1 -> {c5, m3}
  - CATROOM.3 MOUSEROOM.2 -> {c5, c7a, m2}
  - CATROOM.3 MOUSEROOM.4 -> c7a
  - CATROOM.4 MOUSEROOM.0 -> m1
  - CATROOM.4 MOUSEROOM.1 -> {c6, m3}
  - CATROOM.4 MOUSEROOM.2 -> {c6, m2}

**Plan extraction**
- Label states in $F^*$ with fluent values
- Reactive Plan computed from set of control states $S$.
- Control state – has outgoing transition labelled with control.
- Stable state – all outgoing transitions are controls – environment can make no moves – quiescent.

**General Goals**
- General synthesis problem is 2EXPTIME in length of LTL formula.
  - For Generalised Reactivity*, problem can be solved in $N^3$, where $N$ is state space size.
  - Large state spaces can be represented symbolically using BDDs

---

*Synthesis of Reactive(?) Designs, Piterman, Pnueli and Sa'ar, 2004*
Goal Management – Research Challenges

- Specification of domain model and goals
  - application goals
  - system goals
  - covering structure, behaviour, performance …
  - partial knowledge
- Goal refinement
- Runtime Goal & Constraint Checking
- Planning
  - Liveness goals
  - Scalability → Hierarchical Decomposition

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

Generating Revised Plans

- Plan revision through model revision using observations and probabilistic machine learning

Overall SE Research Challenges

- Challenge is to automate and run on-line what are currently off-line RE/design processes e.g. goal-refinement….
- 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.

with Daniel Sykes, Alessandra Russo, Katsumi Inoue and Dominico Corapi
Chapter 9. In conclusion... Model Based Design

Model-based Design and Software tools

Automated software tools are essential to support software engineers in the design process.

Techniques which are not amenable to automation are unlikely to survive in practice.

Extensive experience in teaching the approach to both undergraduates and postgraduates in courses on Concurrency.

Experience with R&D teams in industry (BT, Philips, NATS)
Related Work –

Lots and lots and lots……collaborative teams
multidisciplinary

Daniel Sykes
Alessandra Russo
Will Heaven
Jeff Magee
Sebastian Uchitel
Nicholas D’Ippolito
Victor Braberman
Katsumi Inoue
Andrew McVeigh
Dominico Corapi
Dalal Alrajeh
Axel van Lamsweerde

Current work

- Adaptive autonomous systems
- Model Checking & Machine Learning for requirements elaboration
- Model revision using observations and probabilistic machine learning
- New reference architectures: MORPH

Emphasis on lightweight, accessible and interactive tools tailored for engineers.

LTSA available from: http://www.doc.ic.ac.uk/~jnm/book/

---

Model-based design and analysis of concurrent and adaptive software

Jeff Kramer

Imperial College
London

GSSI - July 2016