Models@run.time for Self-adaptive Reactive Systems: A Controller Synthesis-based Approach

Kenji Tei
Associate Professor
National Institute of Informatics, Japan

Joint work with Moeka Tanabe, Ezequiel Castellano, Leandro Nahabedian, Nicolas D’Ippolito, Shinichi Honiden

tei@nii.ac.jp
http://researchmap.jp/teikenji/?lang=english
How does the system adapt to changes?

Systems face **changes** at runtime

- Cloud
- Robots
- Mobile
- Internet of Things

How do we ensure correctness of software system?
Assurance at Development Time

Kind of requirements guaranteed depends on model and method adopted.
Assurance at Development Time

controller (software)

control

monitor

environment

goal

C

E

G

\[ \text{goal} \Downarrow
\]

\[ \text{p1} \leftrightarrow \text{p2} \]

\[ \vdash \]

\[ \text{C} = \text{E} \]
Environment Modeling for Reactive System

Environment

Controllable actions:
- moveToW
- moveToE
- pickup
- putdown

Uncontrollable (monitorable) actions:
- arriveAtW
- arriveAtM
- arriveAtE
- pickupSuccess
- pickupFail
- putSuccess
- putdFail
Environment Modeling for Reactive System

\[ || E = (\text{MAP} | | W\_\text{ROBOT}). \]

\[
\text{MAP} = (\text{arrive}'w' \rightarrow \text{MAP}'w'), \\
\text{MAP}'w' = (\text{move}'e' \rightarrow \text{arrive}'m' \rightarrow \text{MAP}'m') \\
| \text{move}'w' \rightarrow \text{arrive}'w' \rightarrow \text{MAP}'w' \\
| \text{putdown} \rightarrow \text{putsuccess} \rightarrow \text{MAP}'w' \\
| \text{pickup} \rightarrow \text{pickupfail} \rightarrow \text{MAP}'w'). \\
\text{MAP}'m' = (\text{move}'e' \rightarrow \text{arrive}'e' \rightarrow \text{MAP}'e') \\
| \text{move}'w' \rightarrow \text{arrive}'w' \rightarrow \text{MAP}'w' \\
| \text{putdown} \rightarrow \text{putfail} \rightarrow \text{MAP}'m' \\
| \text{pickup} \rightarrow \text{pickupfail} \rightarrow \text{MAP}'m'). \\
\text{MAP}'e' = (\text{move}'e' \rightarrow \text{arrive}'e' \rightarrow \text{MAP}'e') \\
| \text{move}'w' \rightarrow \text{arrive}'m' \rightarrow \text{MAP}'m' \\
| \text{putdown} \rightarrow \text{putfail} \rightarrow \text{MAP}'e' \\
| \text{pickup} \rightarrow \text{pickupsuccess} \rightarrow \text{MAP}'e').
\]

\[
\text{W}\_\text{ROBOT} = (\text{arrive}'w' \rightarrow \text{ROBOT}), \\
\text{ROBOT} = (\text{move}[\text{Direction}] \rightarrow \text{arrive}[\text{Locations}] \rightarrow \text{ROBOT} \\
| \text{pickup} \rightarrow (\text{pickupsuccess} \rightarrow \text{ROBOT} | \text{pickupfail} \rightarrow \text{ROBOT}) \\
| \text{putdown} \rightarrow (\text{putsuccess} \rightarrow \text{ROBOT} | \text{putfail} \rightarrow \text{ROBOT}) \\
| \text{ended} \rightarrow \text{reset} \rightarrow \text{ROBOT}).
\]
Assurance at Development Time

controller (software) → control

controller (software) → monitor

environment → goal

C \|\| E \|\| G

[[] p1 <> p2 ...]
I Won!
**Motivation**

System *may no longer work*, or *may continue*, but *without any assurances*
Environment is Uncertain

- Sudden increase of user traffic
- Location change
- Security attack
- Slippy floor
- User
- Machine
- Sensor/Actuator
- Disconnection
- Mulfunction
- Unstable performance
- Cloud / External Service
- Service down
- Obstacles

Physical entity
Assuming More Realistic Environment

... MAP['w']=( move['e'] -> arrive['m'] -> MAP['m']
| move['w'] -> arrive['w'] -> MAP['w']
| putdown -> putsuccess -> MAP['w']
| pickup -> pickupfail -> MAP['w'] ),
...

... MAP['w']=( move['e'] -> (arrive['m'] -> MAP['m']
| arrive['w'] -> MAP['w'])
| move['w'] -> arrive['w'] -> MAP['w']
| putdown -> putsuccess -> MAP['w']
| pickup -> pickupfail -> MAP['w'] ),
...
How Much Should We Assume?

- **E\_optimistic**
  - Everything works ideally

- **E\_pessimistic**
  - Everything can go wrong

- **G\_rich**

- **G\_poor**

Risk (low to high) vs. Functionality (poor to rich)
Use models at runtime!
Graceful Degradation by Self-adaptation with Models
Context of My Approach

LTS Controller  LTS Env. Model  LTL Goals

LTL Goals

[] p1
<> p2
Self-adaptation by Models@run.time

Adaptation Engine

1. update env. model
   Analyzer

2. determine req. level
   Planner

3. generate controller
   Executor

4. hot-swap controller
   cached controllers

Motivation

Decision Making
Discrete Controller Synthesis as a Planner

Feedback control for discrete event systems
- driven **not by time** but **rather by events**
- represented as automata, Petri nets, and the like

Synthesize $C$ by solving a control problem $<E,G>$

$(E \ || \ C \models G)$

Synthesis as Two-Player Game
What’s a Game?

A game is composed of an arena and a winning condition.
The winning region for Player 1 is a set of states of the arena in which Player 1 can always win.
A (winning) strategy for Player 1 is defined as a finite number of steps that Player 1 will take to ensure reaching the goal from the initial state, no matter what Player 2 does.

Strategies for reachability are directed-acyclic graphs.
Tool Support

- Discrete Controller Synthesis -

• MTSA (Modal Transition System Analyzer)

http://mtsa.dc.uba.ar
Enact Model

Enactment framework

- Interpret controller model
- Map actions in model to concrete implementation

V. Braberman et al., Controller synthesis: From modelling to enactment, ICSE 2013
Self-adaptation by Models@run.time

Adaptation Engine

1. update env. model

Analyzer

2. determine req. level

G_i

Planner

3. generate controller

Monitor

4. hot-swap controller

Cached controllers

Execution traces

enactment

control

System

Decision Making

Motivation
Self-adaptation by Models@run.time

Adaptation Engine

1. update env. model
2. determine req. level
3. generate controller
4. hot-swap controller

Analyzer

G_i

Planner

cached controllers

Monitor

Enactment

Cached controllers

Situation Awareness

System
Updating LTS-based Environment Model at Runtime

- Environment Model Update -

Updating LTS-based Environment Model at Runtime

Model Update

Consistent

Inconsistent

Situation changes
Updating LTS-based Environment Model at Runtime

Online update should be **accurate** and **efficient**
Proposal - Environment Model Update

Machine Learning Approach

- Environment Model Update -

Existing LTS-model Update

[D.Sykes et.al., ICSE13]

Construct a model with all traces in the window

Use gradient descent algorithm
- repeat computation until convergence

\[
\begin{align*}
    p_{t+1} &= p_t - \eta \nabla MSE(p_t) \\
    P(x_j|B_c) &= \frac{\sum_{b \in B_c \cap b = x_j} \theta_b}{\sum_{b \in B_c} \theta_b} \\
    MSE(p) &= \frac{1}{X_c} \sum_{j=1}^{X_c} (1 - P(x_j|B_c))^2
\end{align*}
\]

Our online LTS-model update

[T, Moeka, SAC16]

Update the model with the latest trace

Use stochastic gradient descent-based algorithm
- the latest data is used for update instead of random picking

\[
MSE(p) = (1 - P(x_j|B_c))^2
\]
- Environment Model Update -

Stochastic Gradient Descent–based
Online Differential Update

At design time

Rules $R$
Threshold $\zeta$

Input

At runtime

Execution Trace

SGD–based Update

$R$ with likelihood

Transform
add transitions whose likelihood exceeds $\zeta$

Execution trace

arrive.e
move.w
arrive.m
move.w
arrive.m

Action Set

<arrive.e, move.w, arrive.m>

<arrive.m, move.w, arrive.m>

<arrive.w1, move.e, {arrive.m1, arrive.w1}>

e.g.

$\langle$pre-condition, action, {post-condition $\alpha, \beta, \gamma, \ldots} \rangle$
Evaluation

Accuracy and settling time

- Error converges quickly

Computational Overhead

- 10,000 ms (10 sec)
- 100,000 times
- 0.1 ms

(GD: Existing method, SGD: Proposal)

Almost the same accuracy
Self-adaptation by Models@run.time

Adaptation Engine

1. update env. model
2. determine req. level
3. generate controller
4. hot-swap controller

Analyzer

Planner

Generator

Cached controllers

Monitor

Executer

execution traces

enactment traces

control

System

Motivation
Other Key Techniques

Ongoing work

1. Environment model learning
   make E neither optimistic nor pessimistic

2. Goal relaxation
   avoid unnecessary degradation

3. Controller synthesis
   generate an assured controller

4. Controller update
   swap controller to new one
Summary

• Context
  – Environment will change at runtime
  – How do we ensure correctness of software?

=> *Models@run.time approach enables decision making when more information is available*

• Tech. Topics
  – How does the system generate a correct controller for unforeseen situation?

=> *Update the environment model and synthesize a correct controller at runtime!*