Surviving in a world of change
Towards evolvable and self-adaptive service-oriented systems

ICSOC 2013

Carlo Ghezzi
Politecnico di Milano
Deep-SE Group @ DEIB
Talk’s roadmap

- Understanding change
- Understanding evolution and adaptation
- How can we detect change?
- How can we detect the need to evolve/adapt?
- How can we react to support evolution/adaptation?
- Lessons learned beyond SASs
The root of the problems: endemic change
The global picture:
the *machine* and the *world* *(Jackson/Zave)*

World (the environment)     Machine

- **Domain properties, assumptions**
- **Shared phenomena**
- **Goals Requirement**
- **Specification**
Domain properties and assumptions

• Both refer to problem world phenomena
• Properties hold regardless of any software-to-be
  - if a positive net force is applied in one direction then the body accelerates in that direction
  - (if plane has touched down then wheels turn)
• Assumptions may be violated
  - submission rate of user requests does not exceed XXX/sec
  - temperature is in the range -40 °C to +40 °C
  - librarians register return of books when users bring borrowed books back
Domain assumptions

May concern

• usage profiles
• users’ responsiveness
• remote servers response time
• network latency
• sensors/actuators behaviors
• …

“Domain assumptions bridge the gap between requirements and specifications”
(M. Jackson & P. Zave)
Dependability arguments

- Assume you have a formal representation for
  - $R =$ requirements
  - $S =$ specification
  - $D = D_p + D_a$ domain properties and assumptions

If $S$ and $D$ are both satisfied and consistent, it is necessary to prove
  - $S, D \models R$
Change

- Requirements change
- Environment changes

- Change is often a manifestation of uncertainty
- Change asks for evolution (of the machine)
Changes may cause evolution

• Changes are exogenous phenomena that may concern
  - \( R \)
  - \( D \) (actually, \( D_a \))

• Changes likely break the dependability argument

• Evolution (of the machine) is a consequence of change
  ▪ we need to change \( S \) (and hence the implementation) to continue to satisfy the dependability argument

\[ S, D \models R \]
Evolution and adaptation

Adaptation is a special case of evolution due to changes in domain assumptions, $D_a$

- an increasingly relevant phenomenon, often due to uncertainty
  - cyber-physical systems
    - interaction with the physical environment
  - user-intensive systems
    - changes in usage profile
  - cloud/service infrastructure
    - platform volatility

Our focus here
On-line vs off-line evolution (type vs instance) vs self-adaptive systems

• Traditionally, response to change is performed off-line by engineers (aka software maintenance)
• More and more often systems are required to be continuously running
• This asks for on-line evolution, i.e. applying changes to the machine as the system is running and providing service
• The special case of self-adaptive systems
  - (instance-level) self-managed on-line adaptation
Self-adaptive system (SaS)

- \( D \) decomposed into \( D_f \) and \( D_c \)
  - \( D_f \) is the fixed/stable part
  - \( D_c \) is the changeable part

- A SaS should
  - **detect changes to** \( D_c \)
  - **modify itself** (the *machine* --- \( S \), and the implementation) to keep satisfying the dependability argument, if necessary

\( S, D \models R \)
Paradigm shift

- SaSs ask for a paradigm shift, which involves both development time (DT) and run time (RT)
- The boundary between DT and RT fades
- Reasoning and reacting capabilities must enrich the RT environment
  - detect change
  - reason about themselves and the possible consequences of change
  - react to change
Models+verification@runtime

• To detect change, we need to monitor the environment.
• The changes must be retrofitted to models of the machine+environment that support reasoning about the dependability argument (a learning step).
• The updated models must be verified to check for violations to the dependability argument.
• In case of a violation, a self-adaptation must be triggered.
Our approach in a nutshell

- **Speciﬁcation** (machine model)
- **Environment** model
- **Self-adaptation**
- **Reasoning**
- **Implementation**
- **Self-adaptation**
- **Monitoring**
- **Execution**

- **Reqs**
- **Development time**
- **Run time**
- **Env**

- **Monitoring**
- **Environment**
- **Speciﬁcation** (machine model)
- **Implementation**
- **Self-adaptation**
• I. Epifani, C. Ghezzi, R. Mirandola, G. Tamburrelli, "Model Evolution by Run-Time Parameter Adaptation”, ICSE 2009
• C. Ghezzi, G.Tamburrelli, "Reasoning on Non Functional Requirements for Integrated Services”, RE 2009
• I. Epifani, C. Ghezzi, G.Tamburrelli, "Change-Point Detection for Black-Box Services”, FSE 2010
Zooming in

• Focus on non-functional requirements
  – reliability, performance, energy consumption, cost, …
• Quantitatively stated in probabilistic terms
• $D_c$ decomposed into $D_u, D_s$
  – $D_u = $ usage profile
  – $D_s = S_1 \land \ldots \land S_n$ $S_i$ assumption on i-th service
Models

- Different models provide different **viewpoints** from which a system can be analyzed
- Focus on **non-functional** properties and quantitative ways to deal with uncertainty
- Use of **Markov models**
  - DTMCs for reliability
  - Reward DTMCs for energy/cost/performance..
- Use of probabilistic model checking for verification that a model satisfies a given property
  - Properties written in PCTL
An example

3 probabilistic requirements:
R1: “Probability of success is > 0.8”
R2: “Probability of a ExpShipping failure for a user recognized as ReturningCustomer < 0.035”
R3: “Probability of an authentication failure is less then < 0.06”
## Assumptions

### User profile domain knowledge

<table>
<thead>
<tr>
<th>$D_{u,n}$</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{u,1}$</td>
<td>$P(\text{User is a RC})$</td>
<td>0.35</td>
</tr>
<tr>
<td>$D_{u,2}$</td>
<td>$P(\text{RC chooses express shipping})$</td>
<td>0.5</td>
</tr>
<tr>
<td>$D_{u,3}$</td>
<td>$P(\text{NC chooses express shipping})$</td>
<td>0.25</td>
</tr>
<tr>
<td>$D_{u,4}$</td>
<td>$P(\text{RC searches again after a buy operation})$</td>
<td>0.2</td>
</tr>
<tr>
<td>$D_{u,5}$</td>
<td>$P(\text{NC searches again after a buy operation})$</td>
<td>0.15</td>
</tr>
</tbody>
</table>

### External service assumptions (reliability)

<table>
<thead>
<tr>
<th>$D_{s,n}$</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{s,1}$</td>
<td>$P(\text{Login})$</td>
<td>0.03</td>
</tr>
<tr>
<td>$D_{s,2}$</td>
<td>$P(\text{Logout})$</td>
<td>0.03</td>
</tr>
<tr>
<td>$D_{s,3}$</td>
<td>$P(\text{NrmShipping})$</td>
<td>0.05</td>
</tr>
<tr>
<td>$D_{s,4}$</td>
<td>$P(\text{ExpShipping})$</td>
<td>0.05</td>
</tr>
<tr>
<td>$D_{s,5}$</td>
<td>$P(\text{CheckOut})$</td>
<td>0.1</td>
</tr>
</tbody>
</table>
DTMC model

R1: “Probability of success is > 0.8” 0.84
R2: “Probability of an ExpShipping failure for a user recognized as ReturningCustomer < 0.035” 0.031
R3: “Probability of an authentication failure is less then < 0.06” 0.056
What happens at run time?

- Actual environment behavior is monitored
- Model updated by using a Bayesian approach to estimate DTMC matrix (posterior) given run time traces and prior transitions
- Boils down to the following updating rule

\[
m_{i,j}(N_i) = \frac{C_i^{(0)}}{C_i^{(0)} + N_i} \times m_{i,j}^{(0)} + \frac{N_i}{C_i^{(0)} + N_i} \times \sum_{h=1}^{d} \frac{N_{i,j}^{(h)}}{N_i}
\]

A-priori Knowledge \hspace{1cm} A-posteriori Knowledge
Model update and failure prediction

- Model checking applied to after each update
- Model checking may *predict* requirements violations
- ... and trigger self-adaptations before violations manifest themselves
In our example

Requirement violated!
Even if no returning customers have been observed

R2: “Probability of an ExpShipping failure for a user recognized as ReturningCustomer < 0.035”
The problem

• Verification subject to (application-dependent) hard real-time requirements
• Running conventional model checking tools after any change impractical in most realistic cases
• But changes are often local, they do not disrupt the entire specification
• Can they be handled in an incremental fashion?
• This requires revisiting model checking algorithms!
Incrementality by parameterization

- Requires anticipation of changing parameters
- The model is partly numeric and partly symbolic
- Evaluation of the verification condition requires partial evaluation (mixed numerical/symbolic processing)
- Result is a formula (polynomial for reachability on DTMCs)
- Evaluation at run time substitutes actual values to symbolic parameters and is very efficient

Working mom paradigm

Cook first. Warm-up later.
Working mom paradigm

Design-Time (offline)

Partial evaluation

Run-Time (online)

<table>
<thead>
<tr>
<th>Trace</th>
<th>Parameter values</th>
</tr>
</thead>
<tbody>
<tr>
<td>p11</td>
<td>0.34</td>
</tr>
<tr>
<td>p12</td>
<td>0.21</td>
</tr>
<tr>
<td>p31</td>
<td>0.12</td>
</tr>
<tr>
<td>p43</td>
<td>0.71</td>
</tr>
<tr>
<td>p32</td>
<td>0.23</td>
</tr>
<tr>
<td>p31</td>
<td>0.54</td>
</tr>
</tbody>
</table>

Analyzable properties: reliability, costs (e.g., energy consumption)

[FormSERA 2012] A. Filieri, C. Ghezzi, "Further steps towards efficient runtime verification: Handling probabilistic cost models"
An example

\[ r = \Pr(s = 5) > r \]

\[ r = \frac{0.85 - 0.85 \cdot x + 0.15 \cdot z - 0.15 \cdot x \cdot z - y \cdot x}{0.85 + 0.15 \cdot z} \]
The WM approach

- Assumes that the Markov model contains absorbing states, and that they are reachable
- Works by symbolic/numeric matrix manipulation
- All of (R) PCTL covered
- Expensive design-time partial evaluation, fast run-time verification
  - symbolic matrix multiplications, but very sparse and normally only few variables
Run-time verification

![Graph showing the relationship between model size and time for different tools (Matlab, Prism, MRMC, WM).]
Further advantage of WM

• Because reachability properties can be expressed via polynomial functions, it is also possible to compute their (partial) derivative and perform sensitivity analysis
  - Which parameters affect most the global quality in the current operation point?
• Similar approach can deal also with rewards
  - Energy consumption, Average Execution time, Outsourcing cost, CPU time, Bandwidth
The rest of the story

• After you detect the need for an adaptation, how do you react?
  • You need to perform a dynamic update
  • This means disconnecting components and ensuring a correct + safe update
  • … but this is subject for another talk
What did we learn?

How/where do we proceed from here?
Run-time management

• The run-time environment for self-adaptive software should not just run applications
  - it should support introspection and reaction
    ▶ on the application’s requirements
    ▶ its behaviour
    ▶ the environment’s behaviour
• Models and continuous verification are essential for introspection and reaction
• But because models change, verification must be efficient
  ✓ constrained by real-time requirements
• This is agility taken to extremes
Beyond self-adaptation

• Lessons learned are far reaching
  - Agile (explorative, incremental) development may become verification-driven by supporting incremental modelling and verification
  - Agility and formal methods may be reconciled rather than being antagonistic

• Vision
  - Towards verification-driven development as complementary to today’s test-driven development
Key feature: incrementality

Incremental verification

Given a system (model) \( S \), and a set of properties \( P \) met by \( S \)
Change = new pair \( S', P' \) where \( S' = S + \Delta S \) and \( P' = P + \Delta P \)

Let \( \Pi \) be the proof of \( S \) against \( P \)
The proof \( \Pi' \) of \( P' \) against \( S' \) can be done by just performing a proof increment \( \Delta \Pi \) such that \( \Pi' = \Pi + \Delta \Pi \)

Expectations:
- \( \Delta \Pi \) easy and efficient to perform
- \( \Delta \Pi \) helps designers reason about change
A long way to go, but possible

• Revisit development models and verification procedures to make them incremental
• Make model-driven development practical
• Package above in IDEs
Acknowledgements

• The work discussed here has been mostly developed thanks to a funding from the European Research Council (Advanced Grant IDEAS-ERC, Project 227977---SMScom)

• ...and thanks to

[Images of individuals]
Thanks for your attention

Questions?