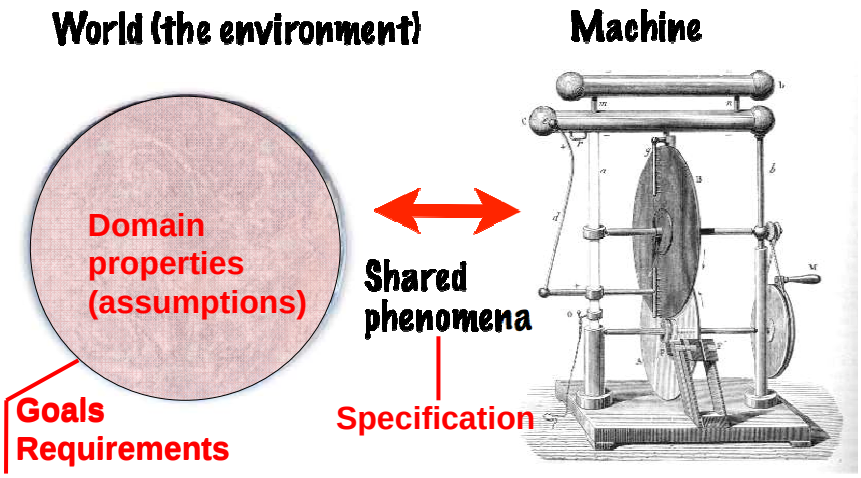
On the use of quantitative models for open-world software

Carlo Ghezzi
Politecnico di Milano
Deep-SE Group @ DEI

Our journey

- What is “open-world” software?
- What makes it hard?
- Non-functional quantitative requirements: why are they crucial?
- Adaptation and evolution
- Modelling and verification: why should they extend to run time?
- Zoom into a current research effort
- Challenges and future work

The *machine* and the *world*



A world of change

- Changes in **goals/requirements**
  - Business level
  - User: skills, profiles (preferences, role, …)
- Changes in **domain**
  - Technology
  - Computational context
    - external components
  - Physical context
    - space, time, …

Multiple ownership

- Systems increasingly built out of parts that are developed, maintained, and even operated by independent parties
- No single stakeholder oversees and controls all parts
- Parts may change over time in an unannounced manner
- Yet by assembling the whole we commit to achieving a certain goal
  - We may even subscribe a *contract (SLA)*

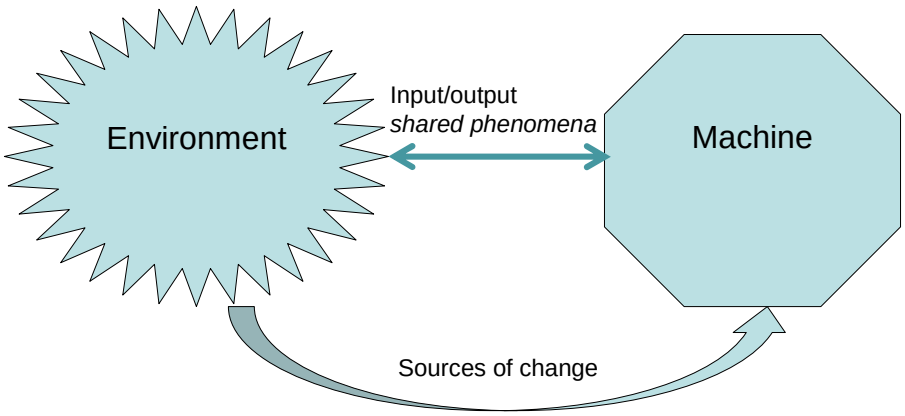
Is this really happening? --business world

- Networked enterprises
  - Business integration infrastructures via Web services are becoming common
- A marketplace for Web services is being created
  - New services created (and possibly exposed) by composing other services
- Networks must be reconfigured to respond to rapidly changing requirements (and changes in business world)

Is this really happening? --ubiquitous computing

- Situational change mainly due to mobility
  - New devices/components encountered/discovered dynamically
  - Interactions/collaborations established dynamically
- Further adaptations due to resource constraints
  - E.g., power consumption
  - Physical conditions (heat, humidity, light, …)

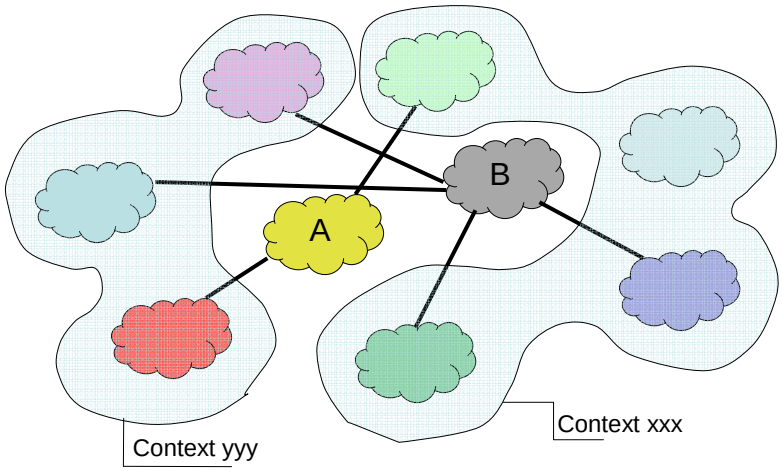
Abstract run time view



The BIG challenge

- Can we support **continuous change and evolution** without compromising **dependability**?
- **Focus on non functional and quantitative requirements**
  - Performance, reliability

What do we need? Flexible composition schemes

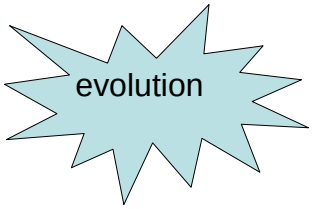


What do we need? Ability to *detect* change

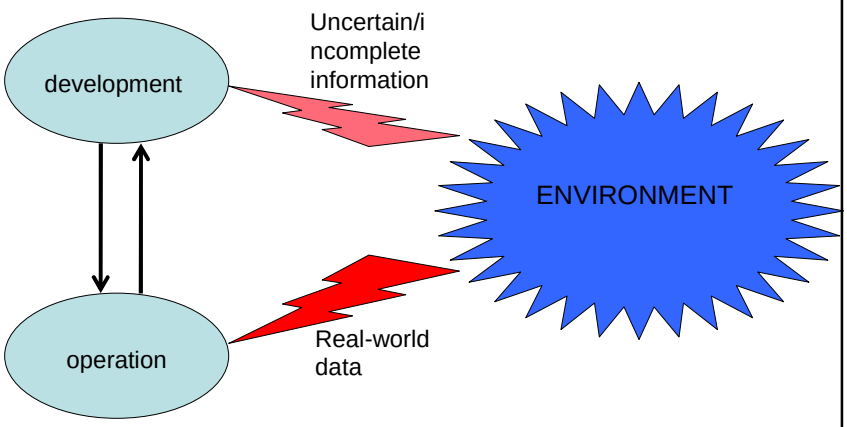
- We need to get real data from the world through (abstract) sensors; e.g., by activating suitable probes
  - ***MONITOR***
- We need to transform data into information
  - ***LEARN***

What do we need? Ability to *react* to change

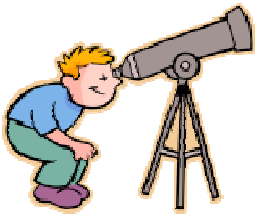
- How can detected changes be used to react by generating a feedback loop to "development" activities?
- Different timescales require different strategies
  - Off-line, with human intervention
    - Re-design/re-deploy/re-run
  - **On-line, self-managed**
    - **A must for perpetual applications**



Development-time/run-time boundary vanishes

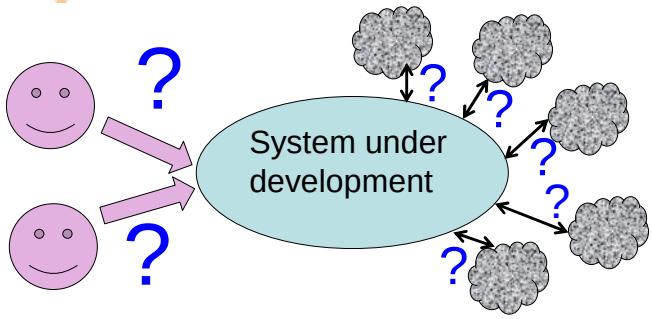


Zoom-in



We explore a seamless development time (DT)/run time (RT) environment, where adaptation is a consequence of uncertainty/changes in the domain

- Input distributions/usage profiles
- External services



Our approach

We build on three key pillars

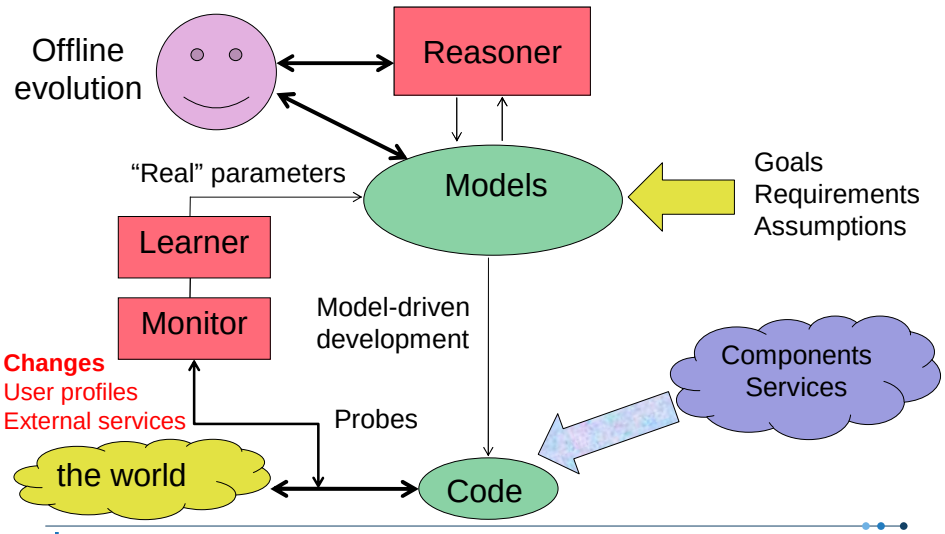
Models Learning

- Markovian models
  - DTMCs
  - CTMCs
- Model checking
  - PRISM
- On going work on using Queuing Networks
- *Open environment should allow adding tools to the workbench*

Further side remarks on models

- Why focus on models in an ephemeral world? Isn't this a contradiction?
  - see anti-model attitude of "agile" methods
- Dependability
  - Models are needed to support systematic reasoning in presence of uncertainty
- Rapid development
  - Implementations may be derived by transformation

Situational adaptive software

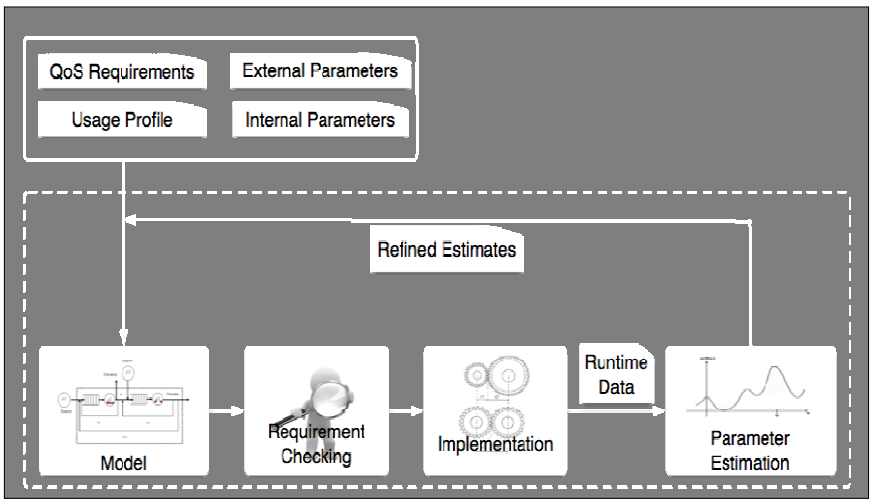


The KAMI system

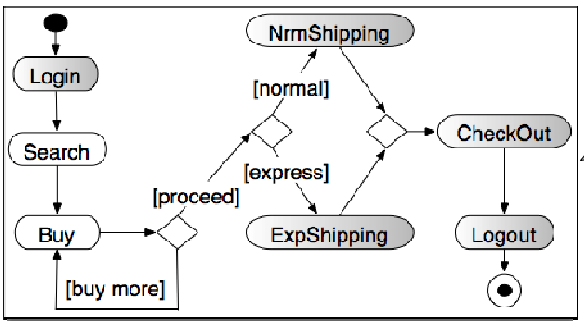
- KAMI: Keep Alive Models with Implementations
- Model adaptation @ run time by learning from monitored data
- Models @ run time for
  - Early discovery/prediction of violations of assumptions made at development time
  - Implementation adaptation

Kami
Keep Alive Models with Implementations

Overall view



KAMI in action: e-commerce service composition



FACT: Users classified as BigSpender or SmallSpender (SS), based on their usage profile.

3 probabilistic requirements:
R1: "Probability of success is > 0.8"
R2: "Probability of a ExpShipping failure for a user recognized as BigSpender < 0.035"
R3: "Probability of an authentication failure is less then < 0.06"

Assumptions

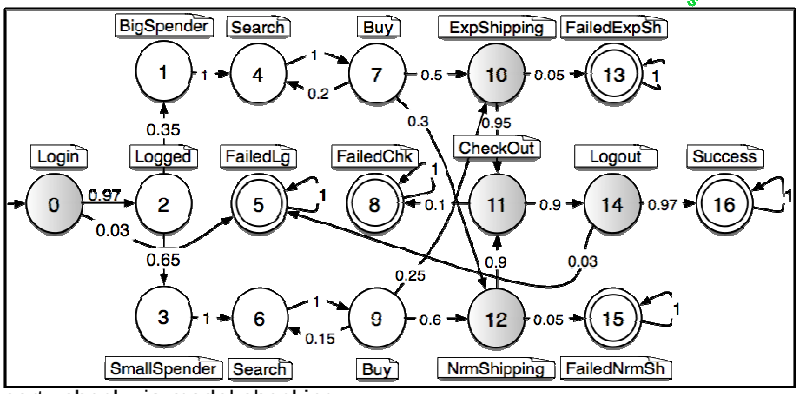
User profile domain knowledge

$D_{u,n}$	Description	Value
$D_{u,1}$	*P(User is a BS)*	0.35
$D_{u,2}$	*P(BS chooses express shipping)*	0.5
$D_{u,3}$	*P(SS chooses express shipping)*	0.25
$D_{u,4}$	*P(BS searches again after a buy operation)*	0.2
$D_{u,5}$	*P(SS searches again after a buy operation)*	0.15

External service assumptions (reliability

$D_{s,n}$	Description	Value
$D_{s,1}$	*P(Login)*	0.03
$D_{s,2}$	*P(Logout)*	0.03
$D_{s,3}$	*P(NrmShipping)*	0.05
$D_{s,4}$	*P(ExpShipping)*	0.05
$D_{s,5}$	*P(CheckOut)*	0.1

DTMC model



Property check via model checking

R1: "Probability of success is > 0.8" 0.084

R2: "Probability of a ExpShipping failure for a user recognized as BigSpender < 0.035" 0.031

R3: "Probability of an authentication failure is less then < 0.06" 0.056

What happens at run time?

- We monitor the actual behavior
- A statistical (Bayesian) approach estimates the updated DTMC matrix (posterior) given run time traces and prior transitions
- Boils down to the following updating rule

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

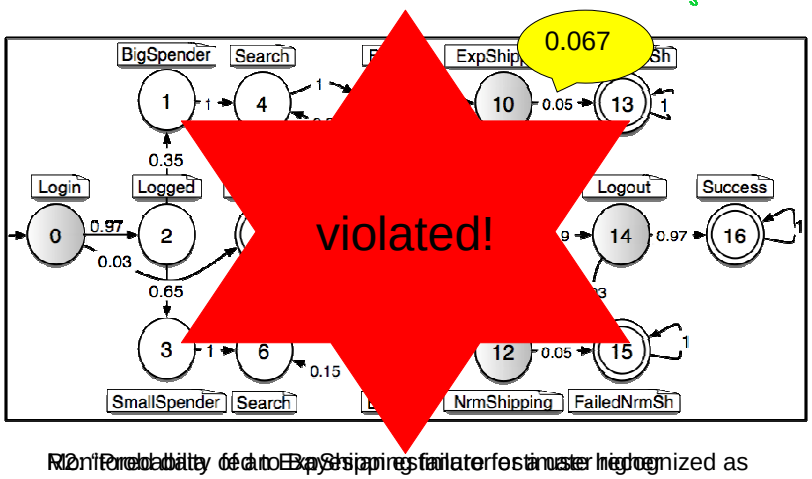
A-priori Knowledge

A-posteriori Knowledge

Why is this useful?

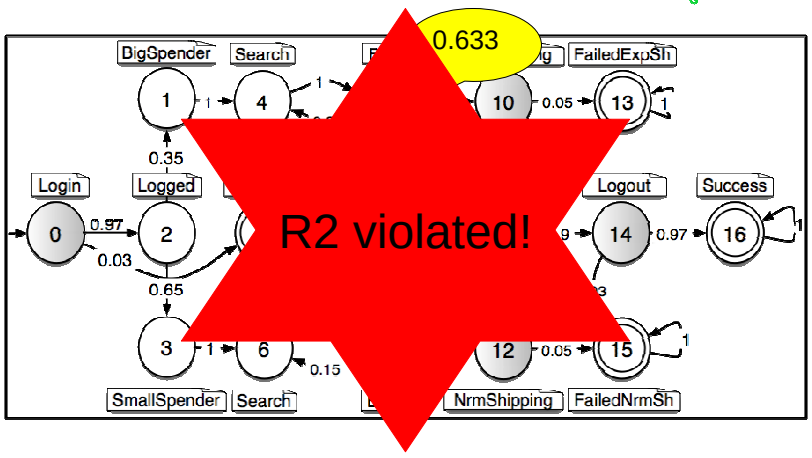
- **Fault**
  - Machine or environment do not behave as expected
- **Failure**
  - Experienced violation of requirement
- Assume that a fault is detected (due to environment). 3 cases are possible
  - All Reqs still valid
    - **OK, but contract violated**
  - Some Req violated + violation experienced in real world
    - **Failure detection**
  - Some Req violated, but violation not experience yet
    - **Failure prediction**

In our example



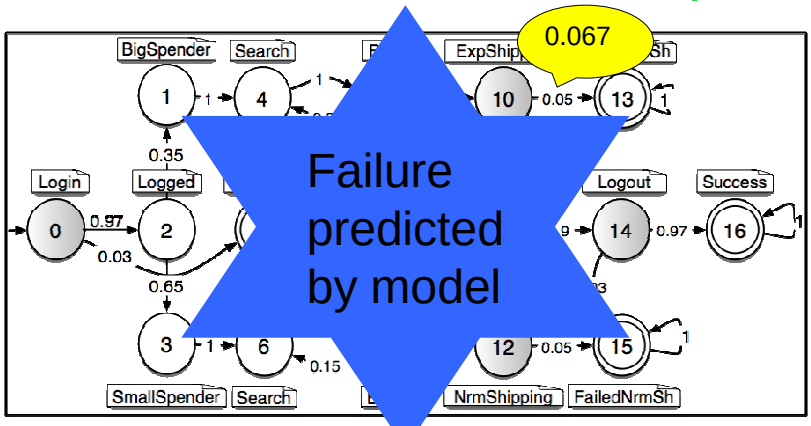
M2 "Probability of a ExpShipping failure for a user higher than Failure probability BigSpender < 0.035"

In our example



Similarly, suppose we detect a change in user profile

In our example



Suppose that execution traces that lead to updating the failure probability of ExpShipping are those involving small spenders

IR2: Probability of a ExpShipping failure for a user recognized as BigSpender < 0.035"

Conclusions

- Modern software systems increasingly live in highly dynamic environments and behave in a situational manner
- Design decision are based on quantitative data and are subject to uncertainty
- Boundary between development time and run time vanishes
- Models should be kept alive at run time and should be adapted to changes in the environment
- Detected changes may trigger model-driven adaptation of the implementation
  - Human-driven, off-line
  - Self managed

On-going and future work

We just scratched the surface, much remains to be done

1. Where do requirements come from? How are they elicited? How do we move from requirements to models?
2. How can a change-point be detected?
3. How can we devise strategies for self-adaptation?
4. Which architectures, middleware, languages are supportive of dynamic change and adaptation?
5. Can we find common realistic case-studies and empirical assessments?
6. How can analysis be done in real time? Incremental analysis techniques?
7. Analysis of partial systems? Inference of specifications?

Thanks to the group: these and many others.....

Acknowledgement

- This work is supported by and Advanced Grant of the European Research Council, Programme IDEAS-ERC, Project 227977---SMScom.

The end

questions?