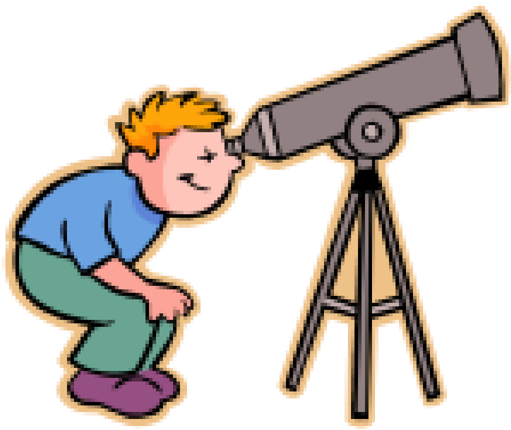
Self Managing Situated Computing



From requirements to specification and (continuous) verification (Part 2)

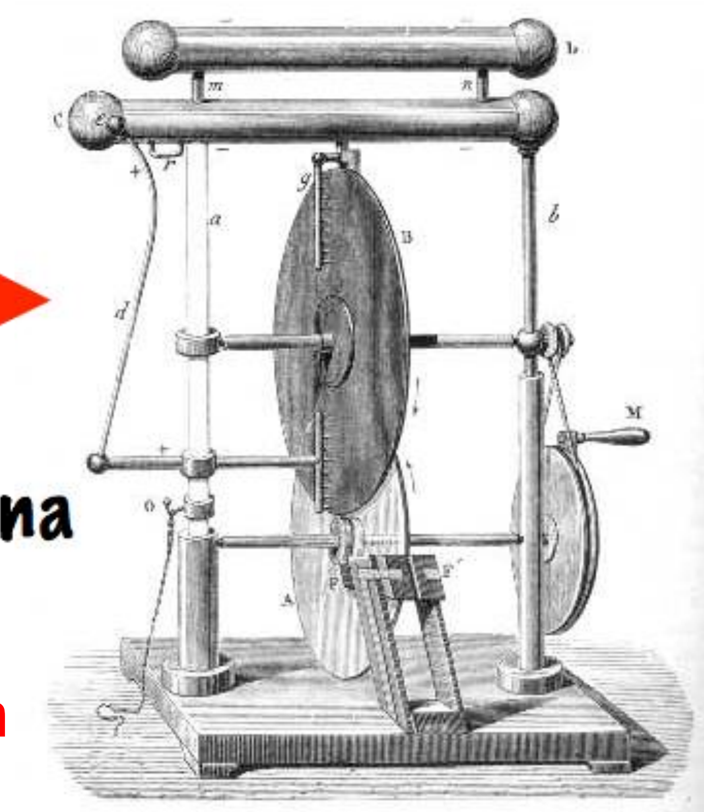
I. Epifani, C. Ghezzi, R. Mirandola, G. Tamburrelli, "Model Evolution by Run-Time Parameter Adaptation", ICSE 2009
C. Ghezzi, G. Tamburrelli, "Reasoning on Non Functional Requirements for Integrated Services", RE 2009.

The *machine* and the *world*

World (the environment)

Machine

Shared phenomena



Goals

Requirements

Specification

Domain assumptions

- Their goal is to bridge the gap between requirements and specifications
- If we have a formal representation as follows
  - R = requirements
  - S = specification
  - D = domain assumptions

  it is necessary to prove that
  - $S \land D \rightarrow R$

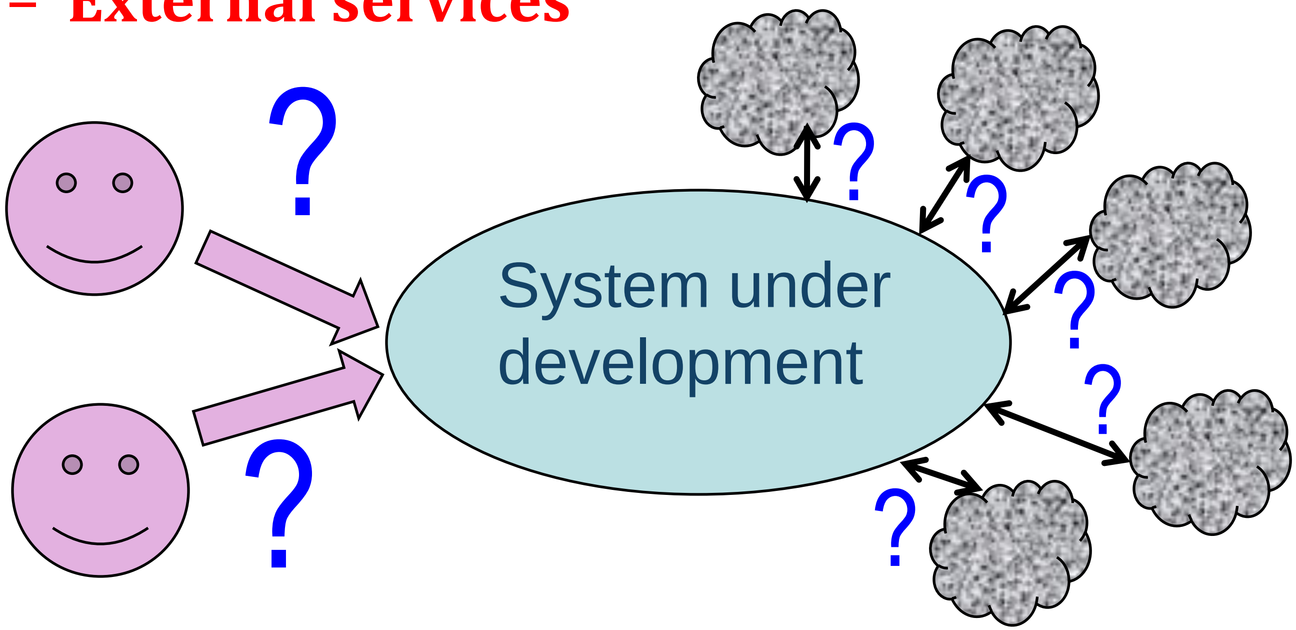
Dependability focus

- Nonfunctional requirements are key aspects of dependability
- We focus here on
  - **reliability**
  - **performance**
- Quantitative assessment necessary
- Uncertainty is a characteristic factor
- Need to deal with **quantitative**, **probabilistic** data

Our setting

At design time there is uncertainty because of incomplete/partial knowledge on the domain + because changes are likely to occur at run-time in

- **Input distributions/usage profiles**
- **External services**



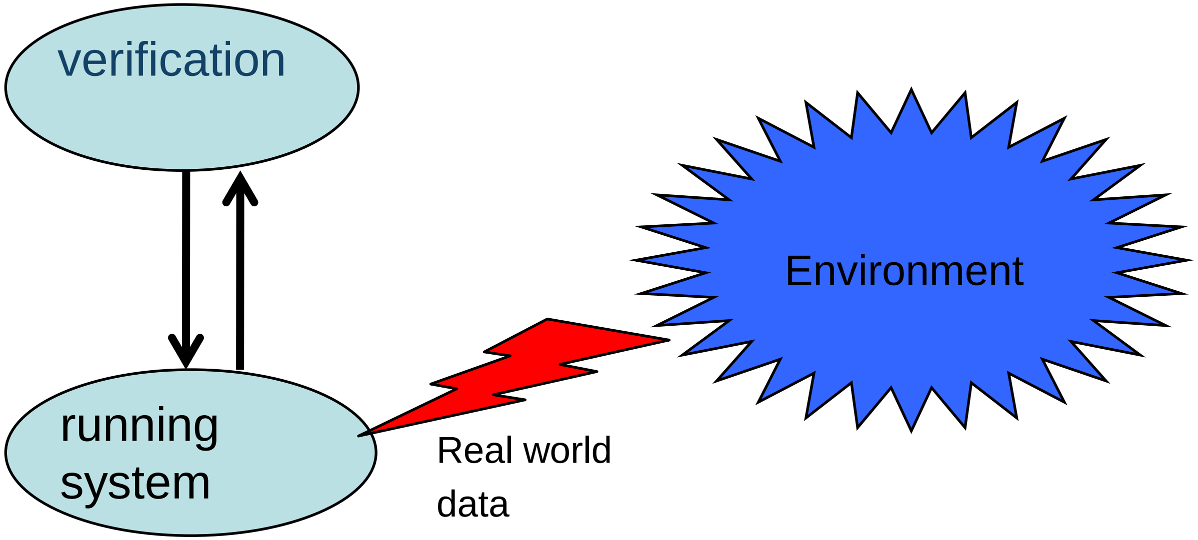
Requirements breakdown

- $D = D_f \wedge D_u \wedge D_s$
  - D_f is the fixed/stable part
  - D_u = Usage profile
  - $D_s = S_1 \wedge .... \wedge S_n$
    - where S_i is the assumption on i-th external service (from SLA document)
- At design-time we need to verify that

$$S \wedge (D_f \wedge D_u \wedge D_s) \rightarrow R$$

At run-time

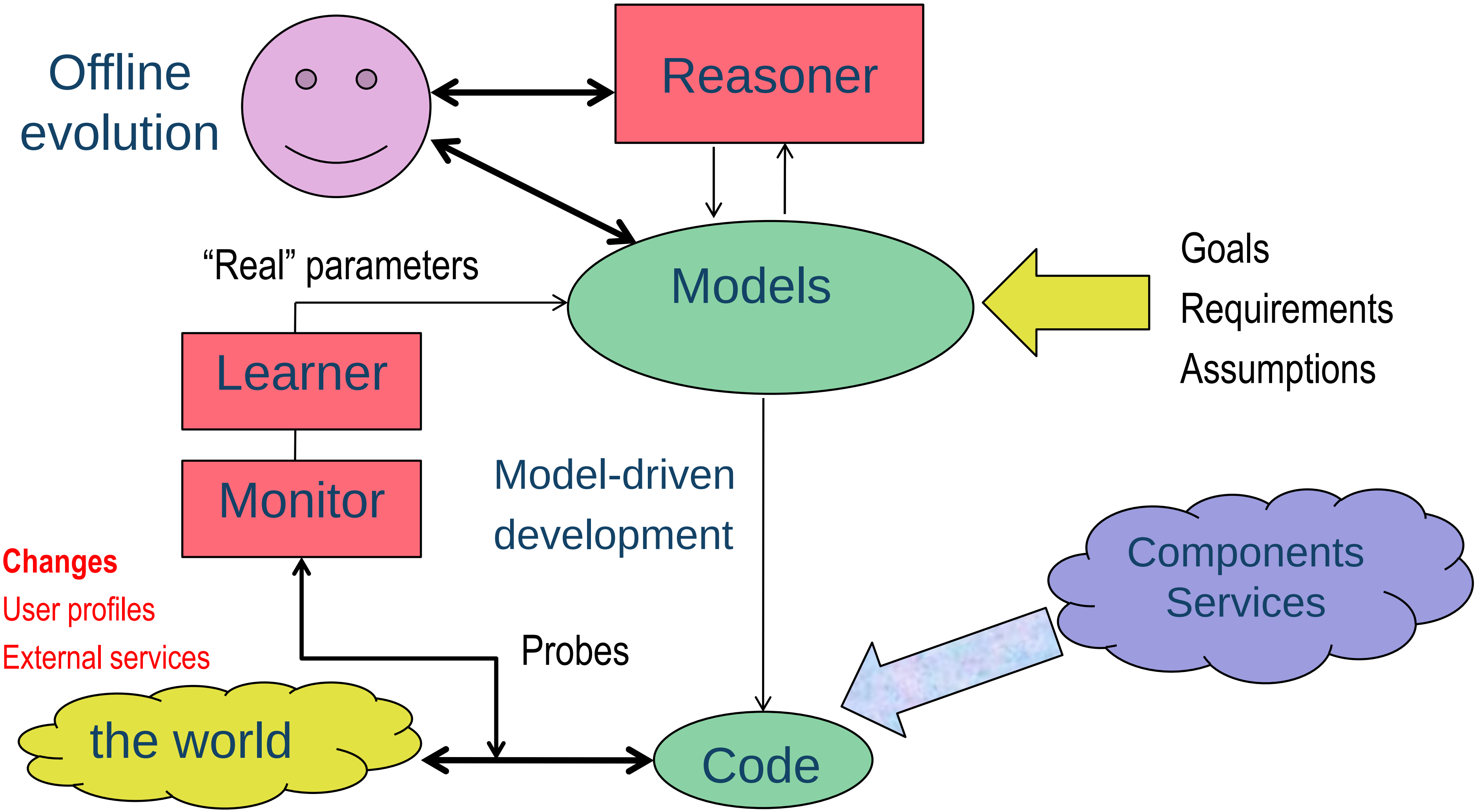
- Reality may subvert our expectations!
- Continuous verification needed



Development-time/run-time boundary vanishes

- The model must be kept alive at run-time and re-analyzed after changes
- Monitored data must be fed back as new parameters of the model
- The mapping data → parameters is achieved via machine learning

Situational adaptive software

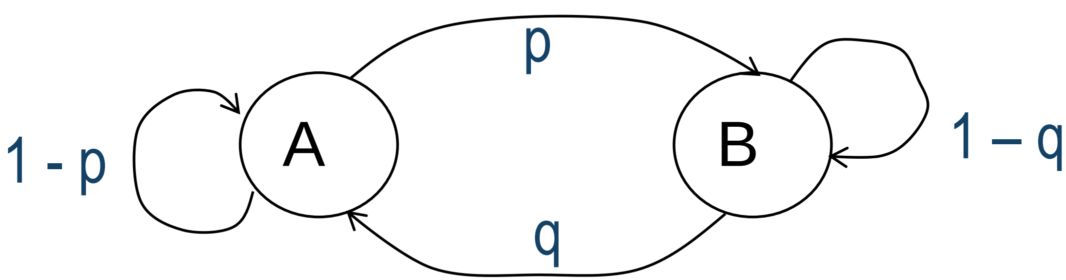


Which models?

- We wish to have modelling notations that allow us to reason about performance and reliability in a quantitative way
- We mostly work with Markov models
  - here we focus on Discrete-Time Markov Chains (DTMCs)

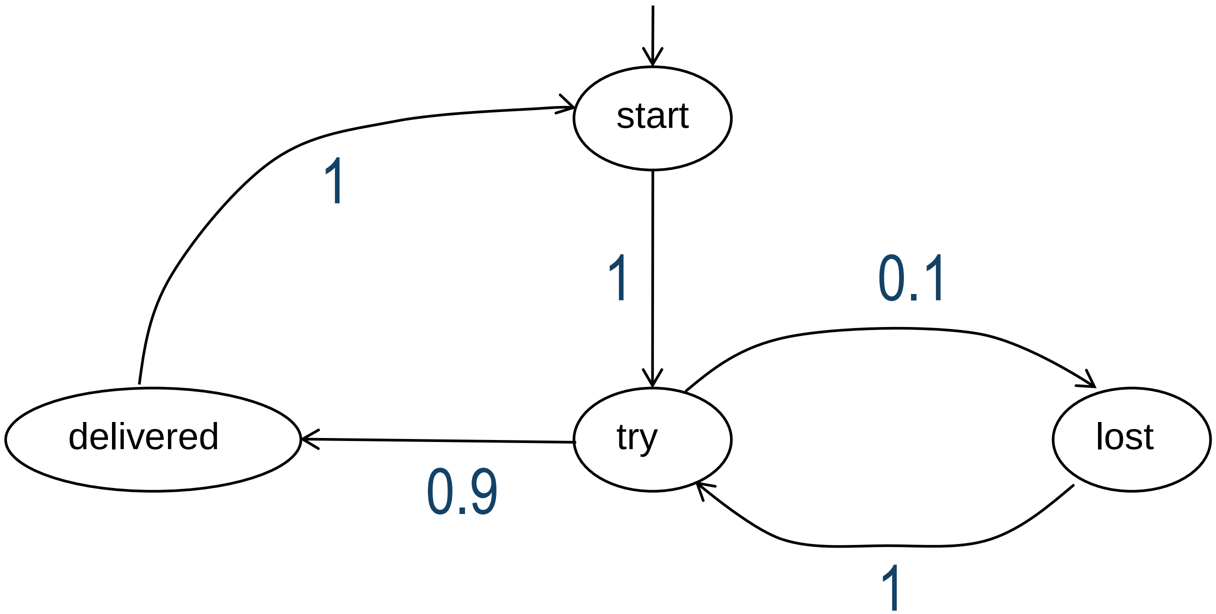
A detour: DTMCs

- A finite-state machine where transitions are labelled with probabilities
  - the sum of probabilities associated with transitions exiting each state is 1
- At every time slot a transition is chosen randomly based on **current** state (a coin is flipped at every time slot)



An example

A simple communication protocol operating with a channel



	S	D	T	L
S	0	0	1	0
D	1	0	0	0
T	0	0.9	0	0.1
L	0	0	1	0

matrix representation

A detour: temporal logic

- We saw a first example of a modal extension to propositional logic: **LTL** (Linear Temporal Logic)
  - it expresses properties over linear sequences of states
  - each state has a **unique** next state
- **CTL** (Computation Tree Logic)
  - can express properties over a branching structure
  - each state can have several next states

CTL

- State formulae
  - $\phi ::= true \mid a \mid \phi_1 \wedge \phi_2 \mid \neg \phi \mid \exists \varphi \mid \forall \varphi$
- Path formulae
  - $\varphi ::= \circ \phi \mid \phi_1 \, U \, \phi_2$

CTL and LTL have incomparable expressiveness

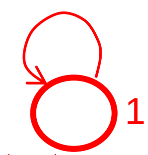
CTL*

- State formulae
  - $\phi ::= true \mid a \mid \phi_1 \wedge \phi_2 \mid \neg \phi \mid \exists \varphi \mid \forall \varphi$
- Path formulae
  - $\varphi ::= \phi \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid \circ \varphi \mid \varphi_1 \, U \, \varphi_2$

CTL* more expressive than LTL and CTL

PCTL

- Probabilistic extension of CTL
- In a **state**, instead of existential and universal quantifiers over paths we can state $P_{\approx p}$ [φ], where p is a probability value and ≈ is <, >, ≤, ≥
  - e.g.: $P_{<0.2}$ [φ] means that the probability for the set of paths (leaving the state) to satisfy φ is less than 0.2
- In addition, **path** formulas also include step-bounded until
  - $\phi_1 \, U^{\leq k} \, \phi_2$
- An example of a reliability statement
  - $P_{>0.8}$ [◊(system state = success)] ← **absorbing state**



PCTL*

- Same philosophy as for CTL* over CTL
  - a path formula can be a state formula

- State formulae
  - $\phi ::= true \mid a \mid \phi_1 \wedge \phi_2 \mid \neg \phi \mid P_{\approx p}[\varphi]$
- Path formulae
  - $\varphi ::= \phi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid o\, \varphi \mid \varphi_1 \, U \, \varphi_2$

PCTL* is more expressive than PCTL

- An example of a PCTL* reliability statement

$P_{constr}[\Diamond(through\_state \wedge \Diamond absorbing\_state)]$

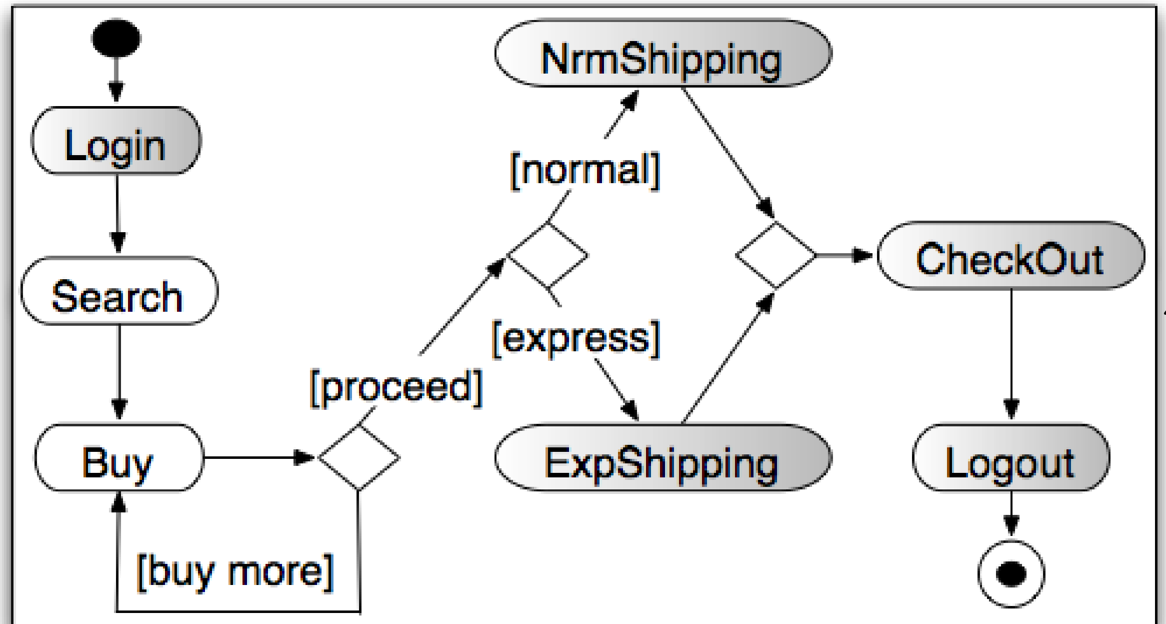
Probabilistic model checking

- Given:
  - a DTMC M
  - a state s of M
  - a PCTL or a PCTL* state formula ϕ

  determine if M |= ϕ
- Results of analysis:
  - OK, property satisfied
  - property violated
  - ... out of memory
- Existing tools
  - PRISM (Kwiatkowska et al.) http://www.prismmodelchecker.org/
  - MRMC (Katoen, Hermanns, ...) http://www.mrmc-tool.org/trac/

C. Baier, JP Katoen, "Principles of model checking" MIT Press, 2008

Our approach (KAMI) in action



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

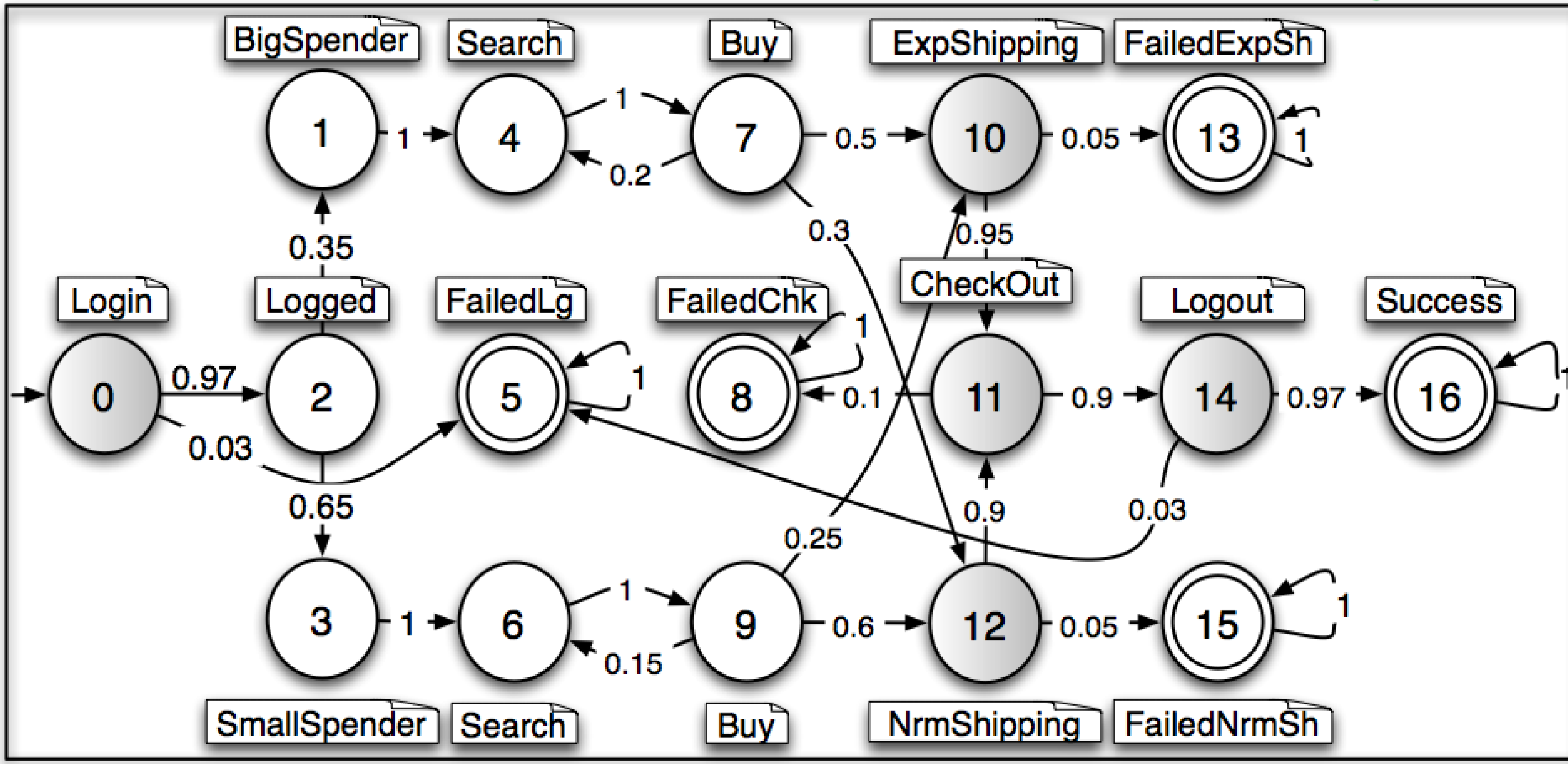
User profile domain knowledge

$D_{u,n}$	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

Self Managing Situated Computing

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

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

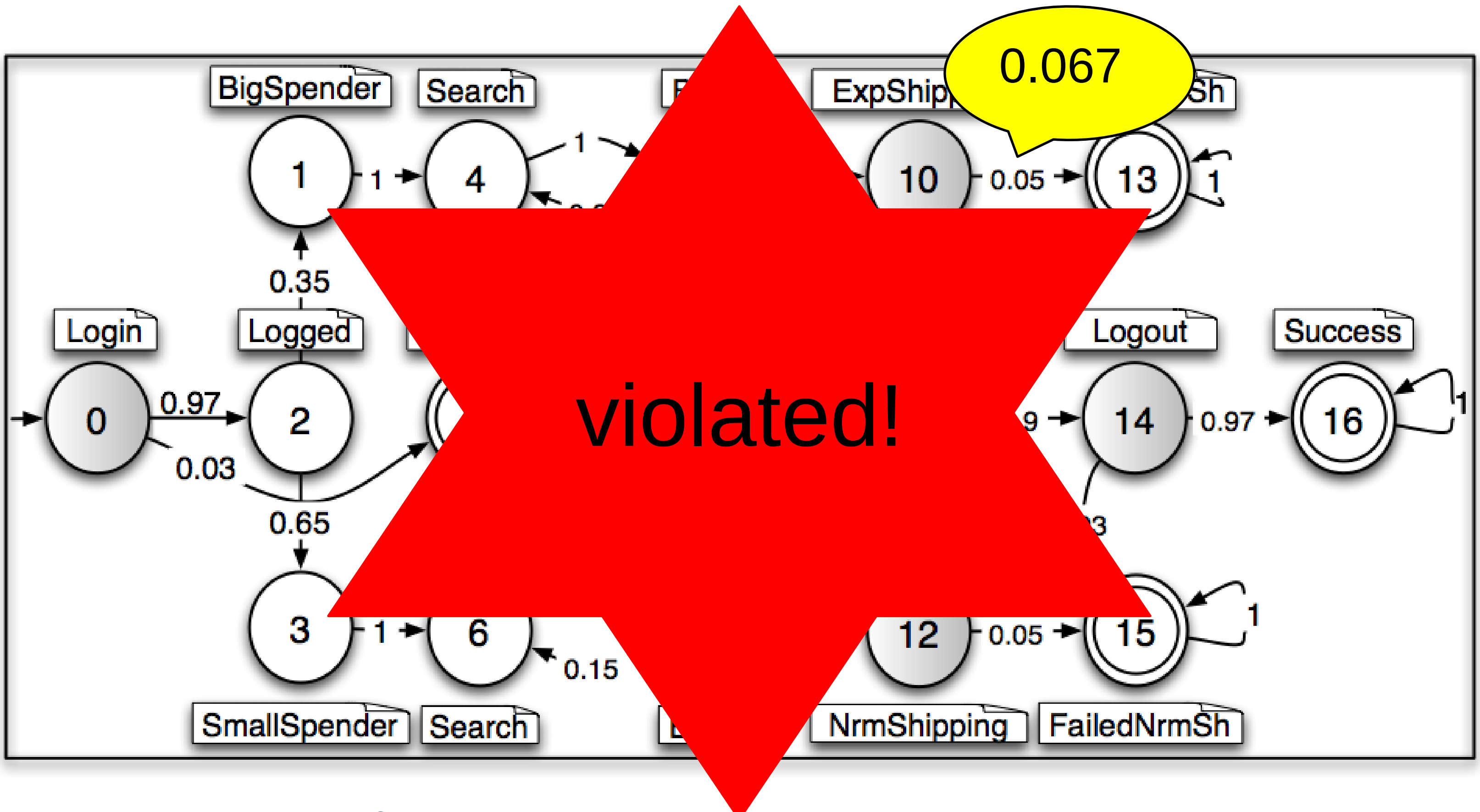
A-priori Knowledge

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**

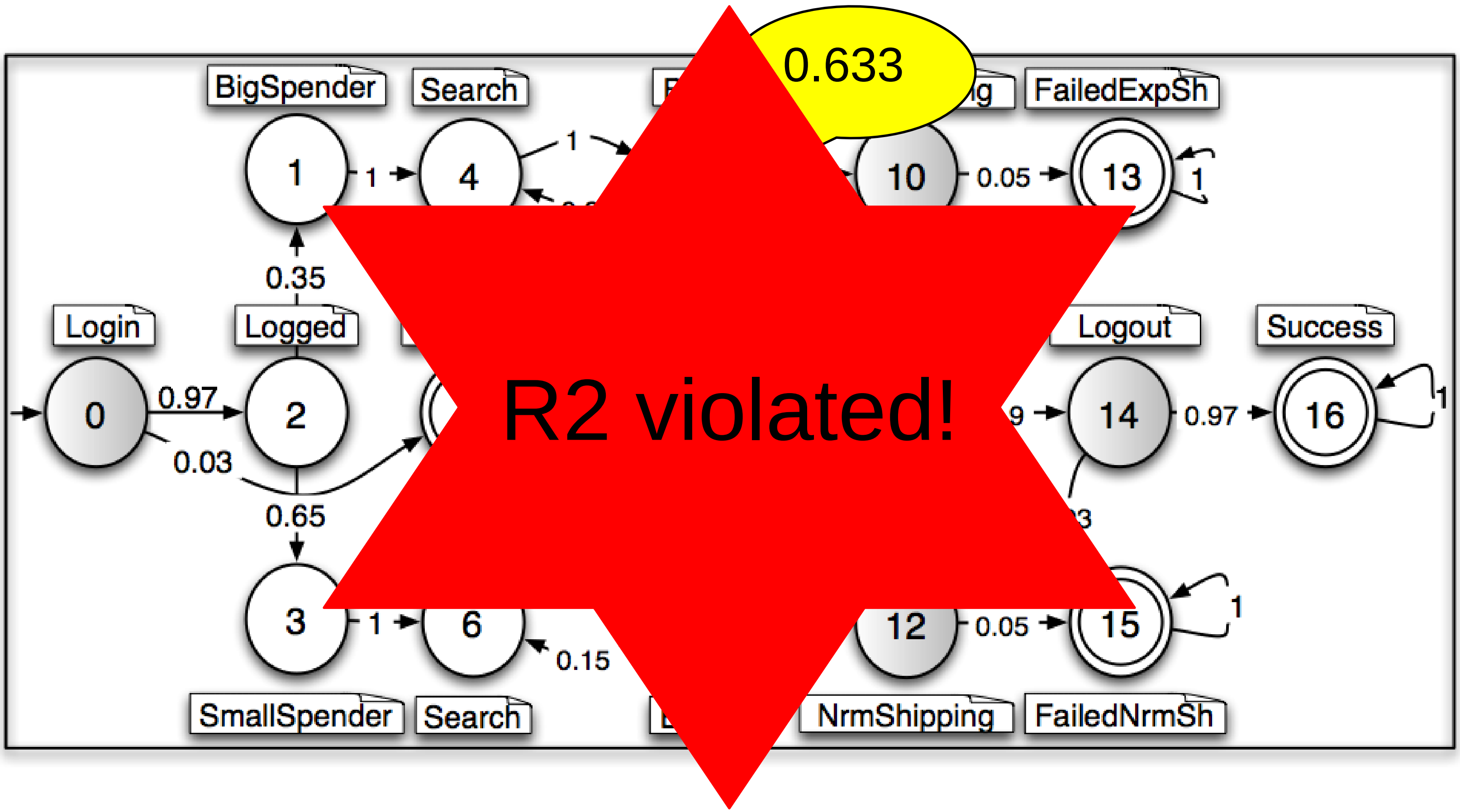
In our example



Monitored data fed to Bayesian estimator: estimate higher failure probability

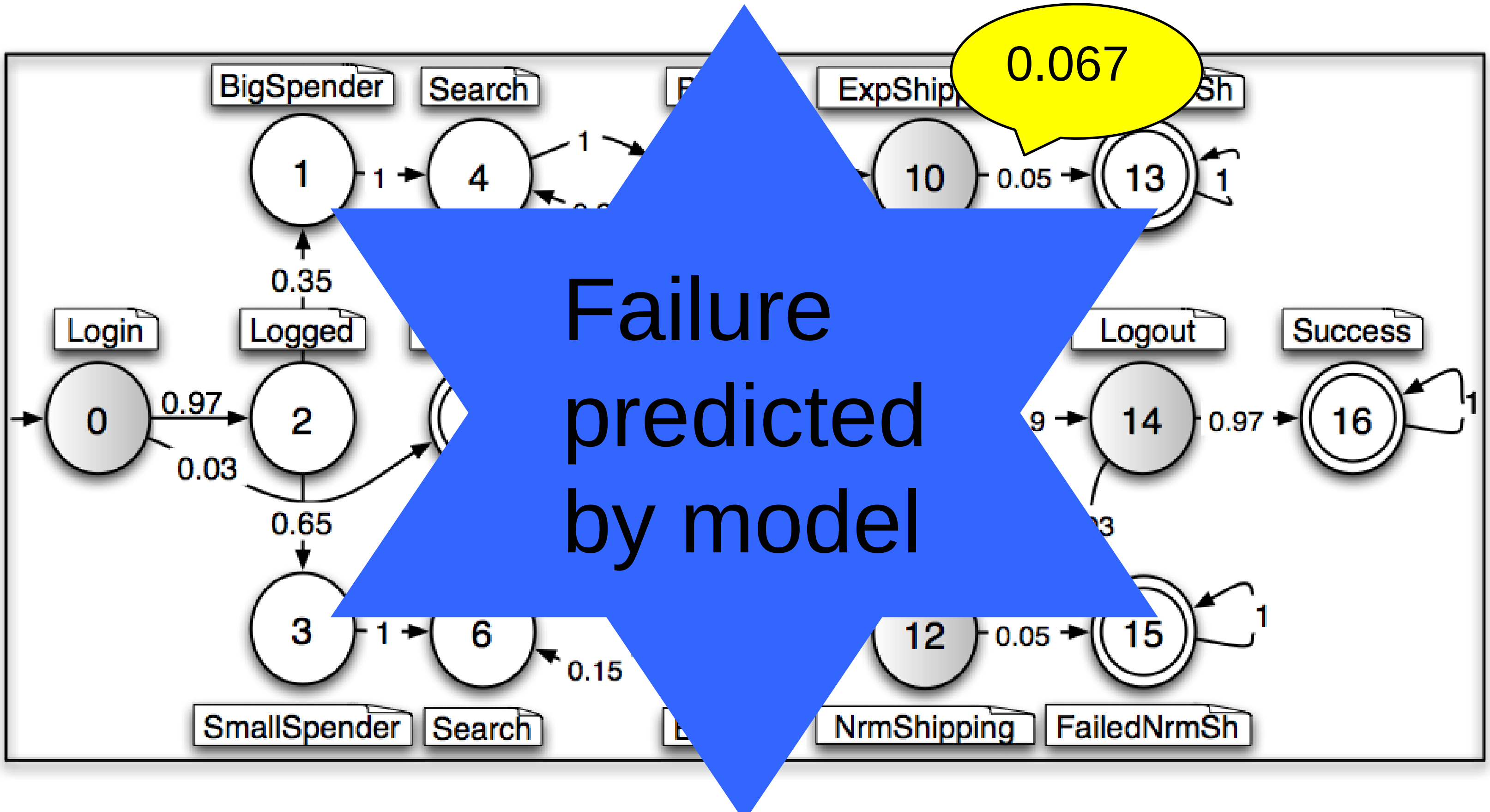
R2: "Probability of an ExpShipping failure for a user recognized as BigSpender < 0.035"

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

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

Development-time vs run-time

- DT and RT verification requirements may differ in terms of time constraints
- Time constraints for RT verification are especially stringent if the results are to be used for adaptation
- Issues
  - can the RT model be the same as the one used at DT?
  - should it be a simplified (less precise) version?
  - can analysis be performed incrementally?

Cost of model checking

- Model checking is an expensive analysis technique
  - PCTL
    - Polynomial in size(DTMC)
    - Linear in size(formula)
  - PCTL*
    - Polynomial in size(DTMC)
    - Double exponential in size(formula)

A possible solution

- Assumptions
  - we know which parts of the model may change (DTMC parameters)
  - the structure does not change
- Then a verification formula can be pre-computed at design-time
  - variables in the formula represent dynamic data, whose values become known at run-time
- Run-time verification can be performed efficiently on-the-fly

Reaction policies

- A largely unexplored territory
- We tried with rebinding policies
- But we are far from a complete picture
- Problem statement
    - many "equivalent" ("substitutable") services exist for any abstract service invocation
    - how can the "best" service be chosen?

C. Ghezzi et al. "QoS Driven Dynamic Binding in-the-many", QoSA 2010, LNCS

Selection problem and load balancing

- Service selection similar to load balancing where
  - services are the resources
  - composite workflows are clients
- Resources are heterogeneous
- Clients must select the resource
  - based on which information?

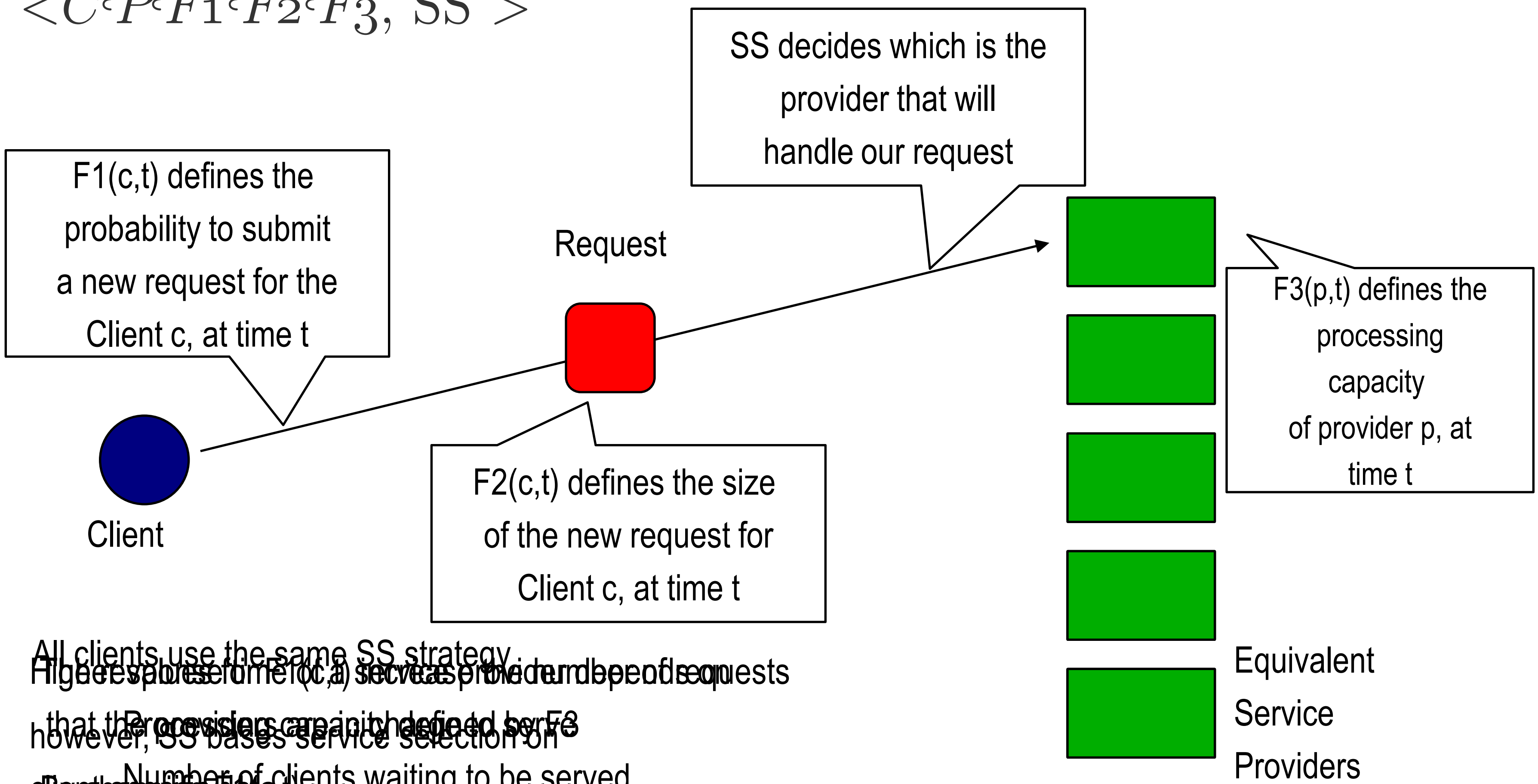
Framework definition

- A multi-client multi-provider stochastic system is a 6-tuple:

  < C, P, F1, F2, F3, SS >
  - C: set of clients
  - P: set of service providers
  - F1: client's probability to submit a service request
  - F2: size of a request
  - F3: provider's processing rate
  - SS: selection strategy

Framework definition

$< C, P, F1, F2, F3, SS >$



All clients use the same SS strategy, however, SS bases service selection on

Processing capacity defined by F3

Number of clients waiting to be served

Higher values for F1(c,t) increase the number of requests that the providers can handle to serve

By changing F1(c,t)

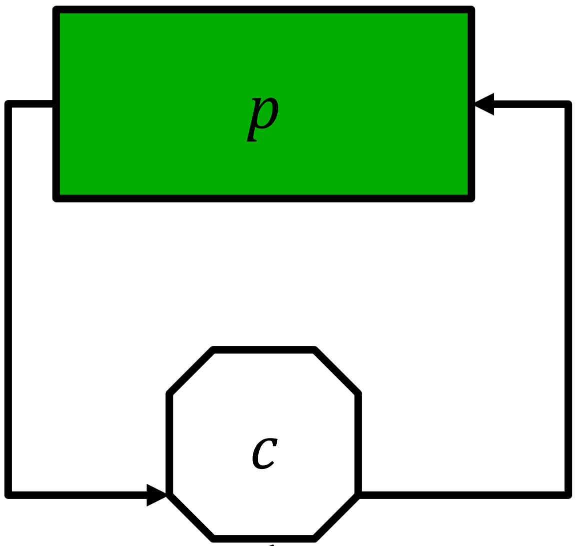
we can stress the system with a variable load

Efficiency estimator

- For each service, each client c stores its knowledge in the efficiency of providers in an estimator vector ee_c
  - $ee_c(p)$ is the current client's evaluation of p's performance

1) How can ee_c be managed?
2) Which provider should be selected based on ee_c?

Updating *ee*



$ee_c(p) := \mathrm{W}\,\mathrm{T} + (1 - \mathrm{W})\, ee_c(p)$

$jd_c(p)$ ++

$\mathrm{W} := w + (1 - w) / jd_c(p)$

- T is the response time measured by the client
- *w* determine the weight of the current record respect to the previous ones
- $jd_c(p)$ collects the number of requests served by provider *p*

Notice that:

- Only the *p* entry is updated.
- The estimate of all other providers **does not change**

Selection Strategies

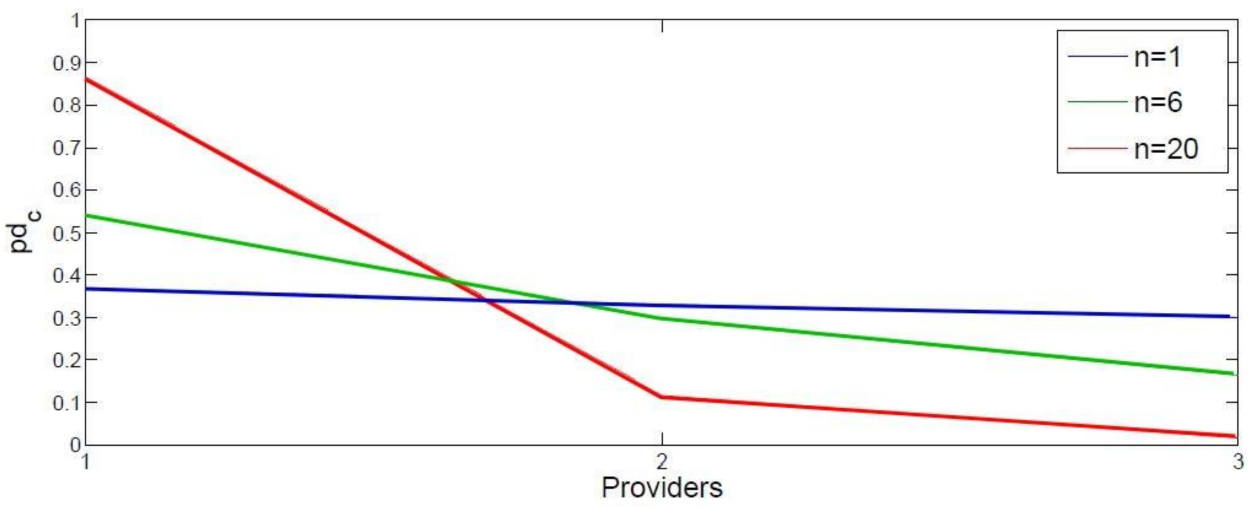
- **Distributed** : each client selects the service according to its own available information
  - Minimum Strategy
  - Probabilistic Strategy
  - Collaborative Strategy
- **PROSS**: choice delegated to a (logically) centralized proxy
  - Proxy Minimum Strategy
  - Proxy Probabilistic Strategy

Minimum Strategy

- Select service provider with the best expected performance
  - minimum value in ee_c
- **Pros**
  - the simplest and most intuitive algorithm
- **Cons**
  - bad load balancing
  - poor efficiency

Probabilistic Strategy

- Select the service provider with probability pd_c.
- pd_c is a function of:
  - ee_c
  - n : how much "*explorative*" is the client

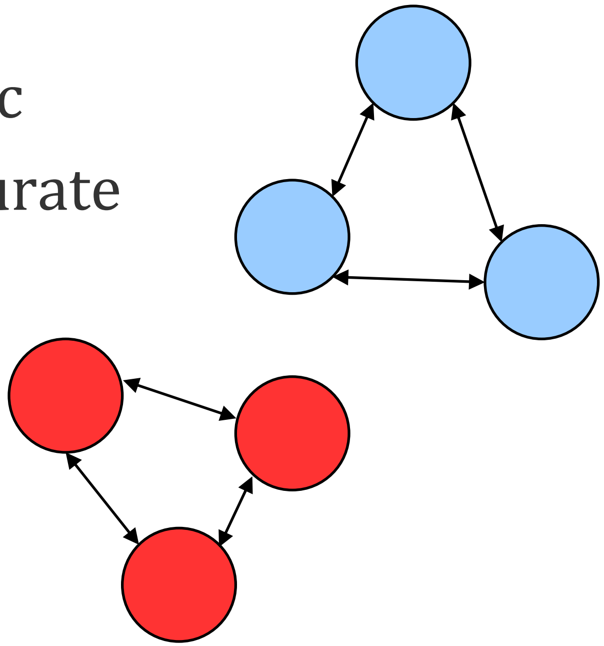


Probabilistic Strategy

- Pros: Minimum Strategy problems solved!
  - Better Load Balancing
  - More efficient!
- Cons: according to the definition of ee_c, performance estimates may not reflect the current situation (they are based on that client's experience only!
- How to solve?
  - Communicating Clients
  - (Logically) Centralized Approach

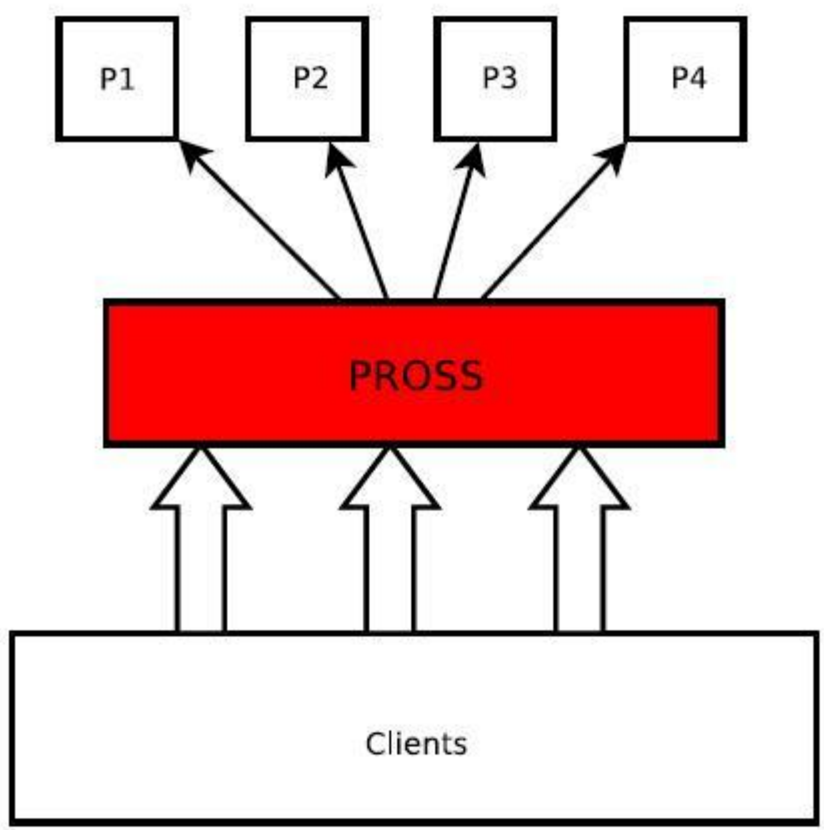
Collaborative Strategy

- Allows communication between clients
- Each client *c* maintains its own ee_c
- *ee* vectors of a **neighborhood** are shared when a decision is made
- Selection still remains probabilistic
- **Pros:** decision based on more accurate performance estimates
- **Cons:** the local communication is not sufficient to obtain good results



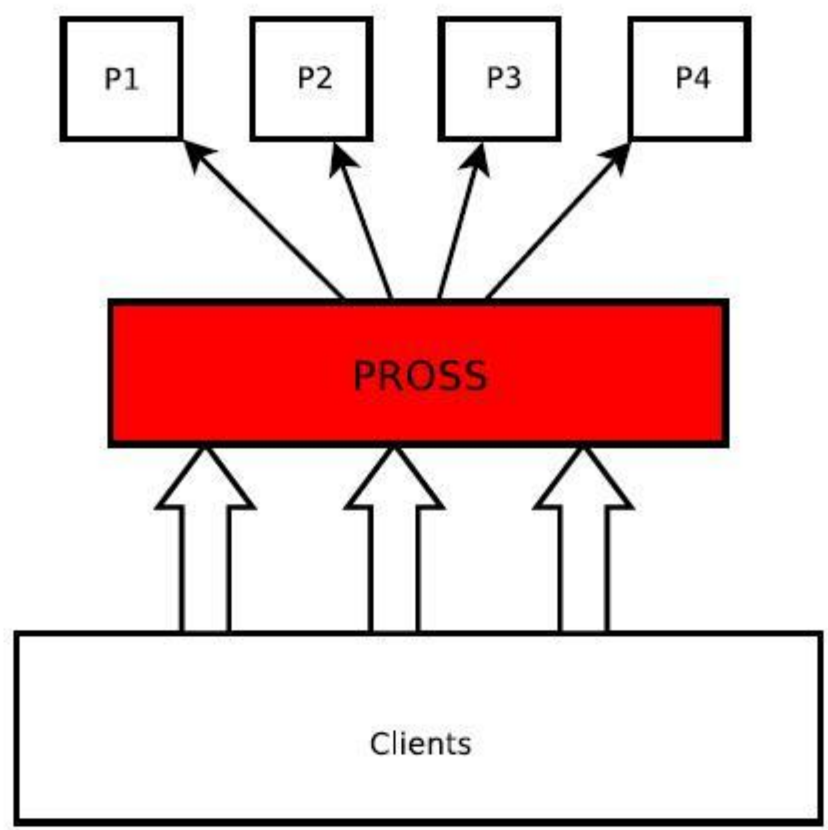
PROSS

- **PRO**xy **S**ervice **S**elector, a centralized entity which:
  - Makes the decision
  - Links to the client submitting the request
- Information available:
  - Global efficiency estimator *ee*
  - *pending_requests*



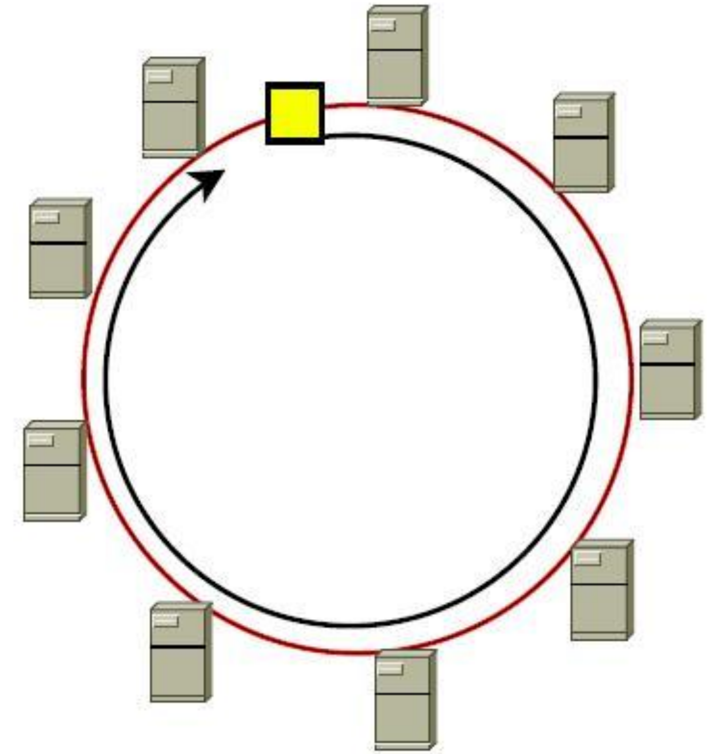
PROSS

- PROSS acts as a load balancer
- Service providers are unaware of the entire selection process
- Service interaction paradigm simplified



A possible distributed PROSS

- Token Ring Architecture
- Global *ee* computed as the **average** of the *ee* vectors of all nodes
- *pending_requests* as the sum of the *pending_requests* vectors of all nodes
- Consistency maintained respect to the logical view



token=[ee ; pending_requests]
ee=[ee(p1),...ee(pi),...ee(pn)]
pending_requests=[pr(p1),...pr(pi),...pr(pn)]

Validation

- Numerical simulations in Matlab of the Multi-client multi-provider stochastic system
- Setup of a number of possible scenarios
  - Different probabilistic request submission function for the pool of clients
  - Different processing capacities for the pool of providers
- Study of the performances of the different SS strategy for each of the scenarios defined previously
- PROSS wins—see (*) for details

(*) C. Ghezzi et al. “QoS Driven Dynamic Binding in-the-many”, QoSA 2010, LNCS