Laboratoire LSR
Logiciels Systèmes Réseaux
Software, Systems, Networks

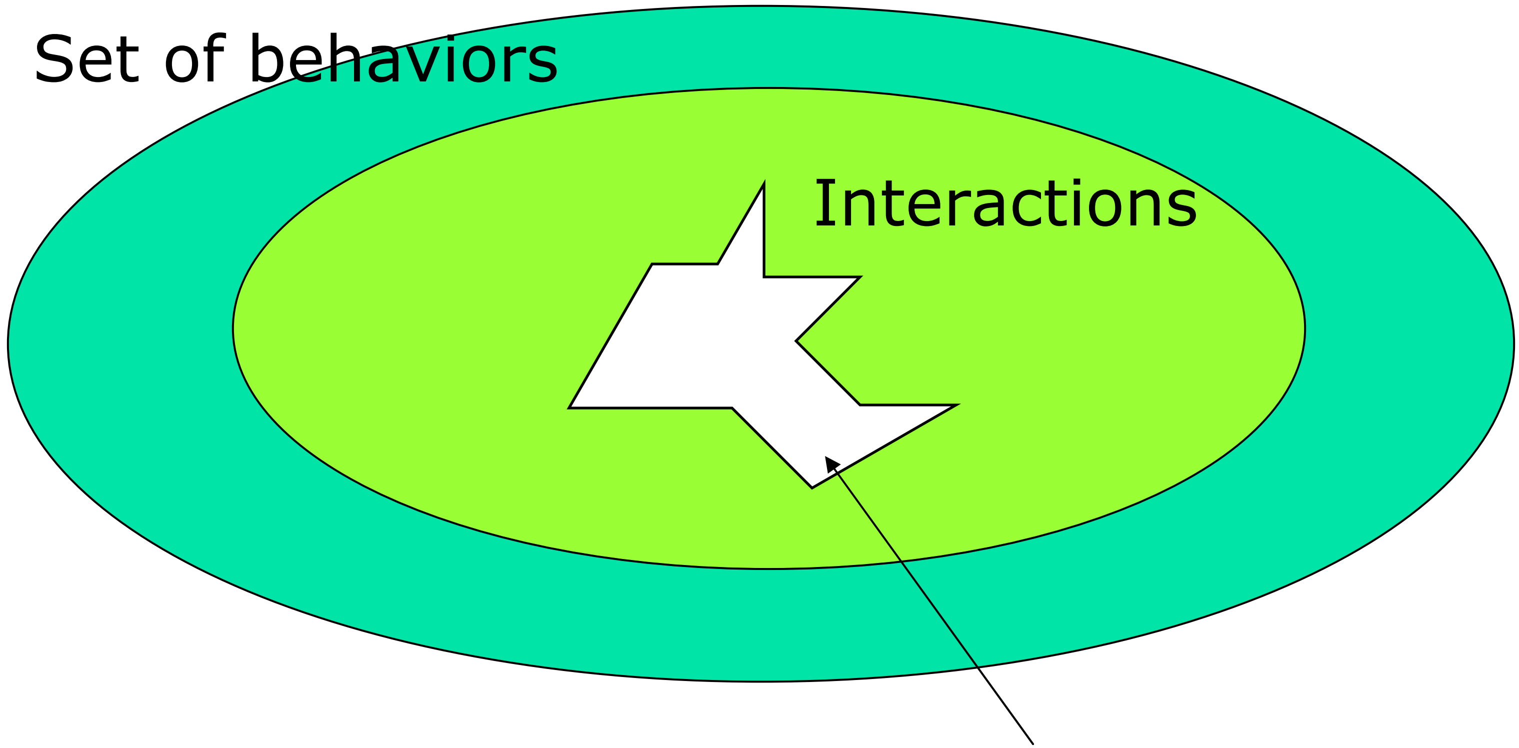
An environment for Interactive Service Specification

K. Berkani, R. Cave, S. Coudert, F. Klay,
P. Le Gall, Farid Ouabdesselam, J.-L. Richier
France Télécom R&D, Lannion, France
LaMI, Université d'Evry, France

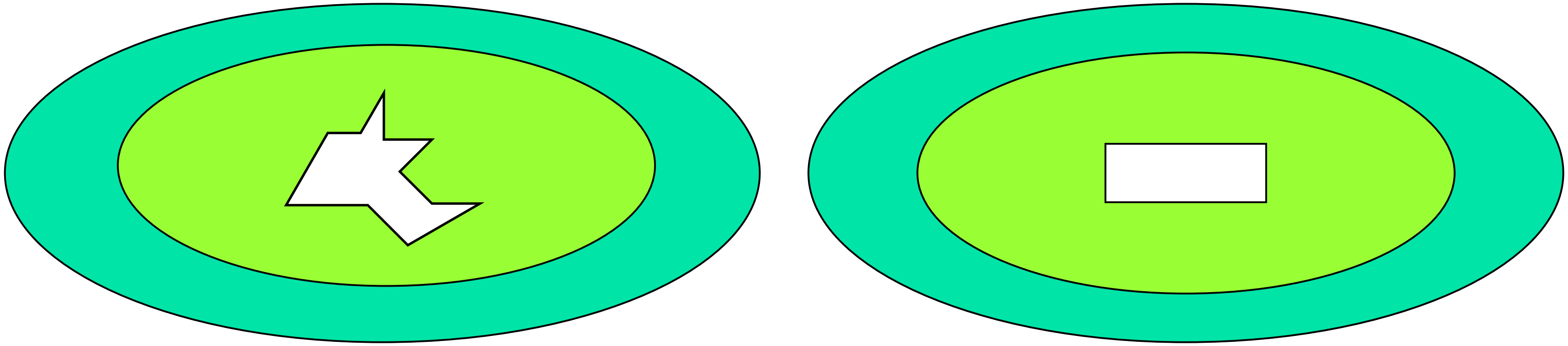
Summary

- Undesired interactions at the requirements level : a subjective notion, efficiency and effectiveness of property-based detection
- A new feature integration method with filtering: composition, static validation, dynamic validation
- Interaction expert-based “fault model”, interaction patterns, automated generation of animation guides towards interaction-prone situations

Singling out undesired feature interactions



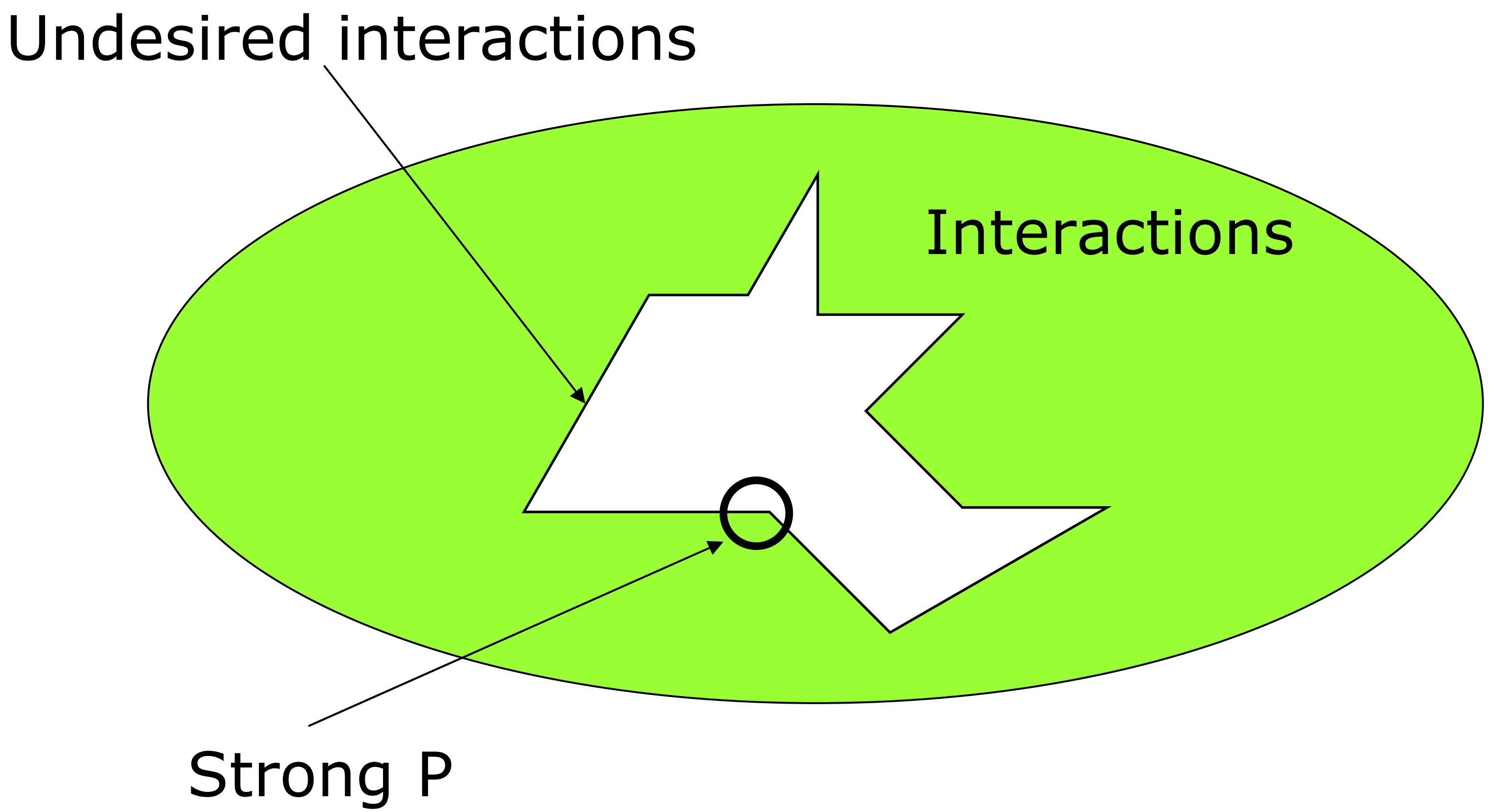
Interactions : a subjective notion



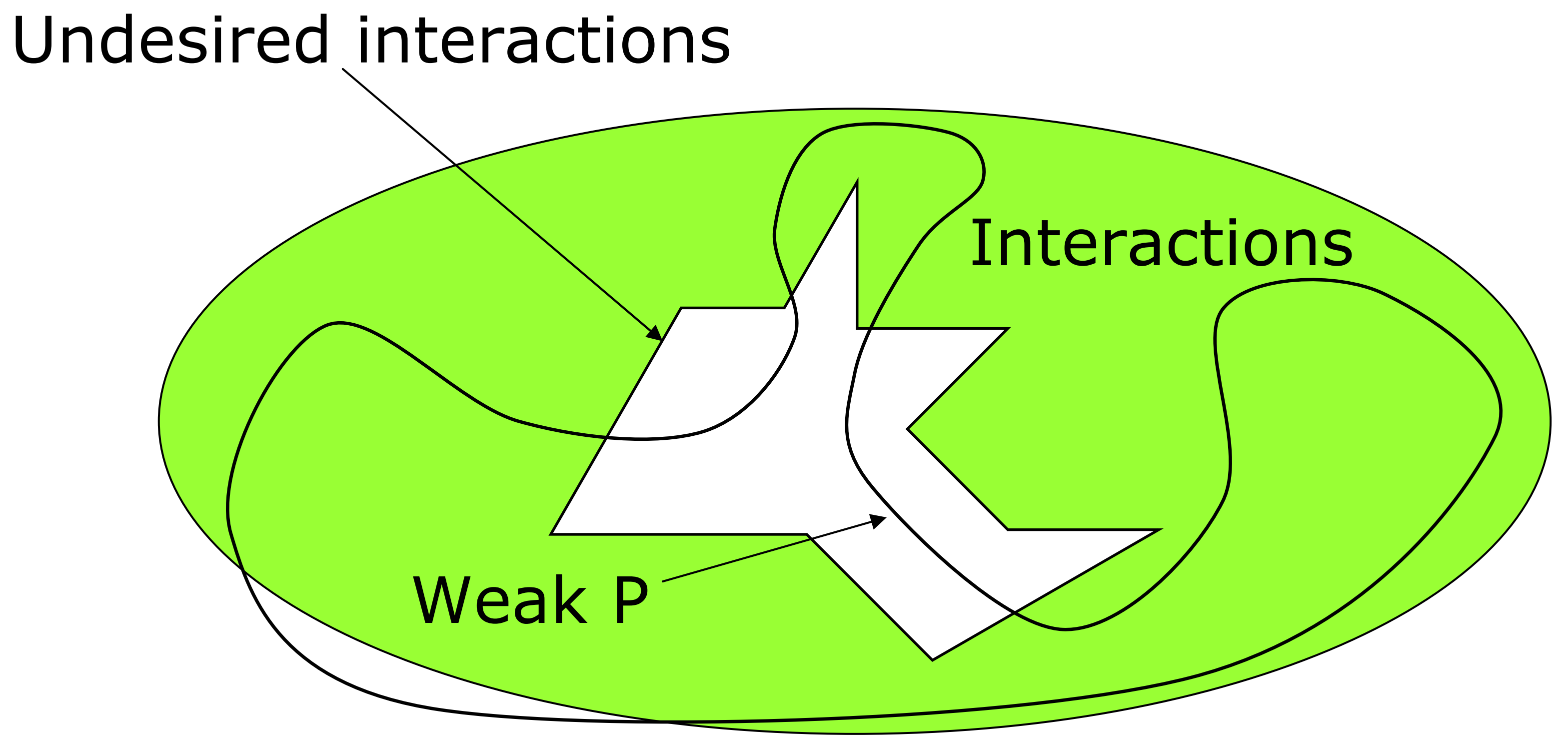
Loose definition of interactions (imprecise, partial and subjective)

Absolute definition of interactions (precise, complete and indeniable)

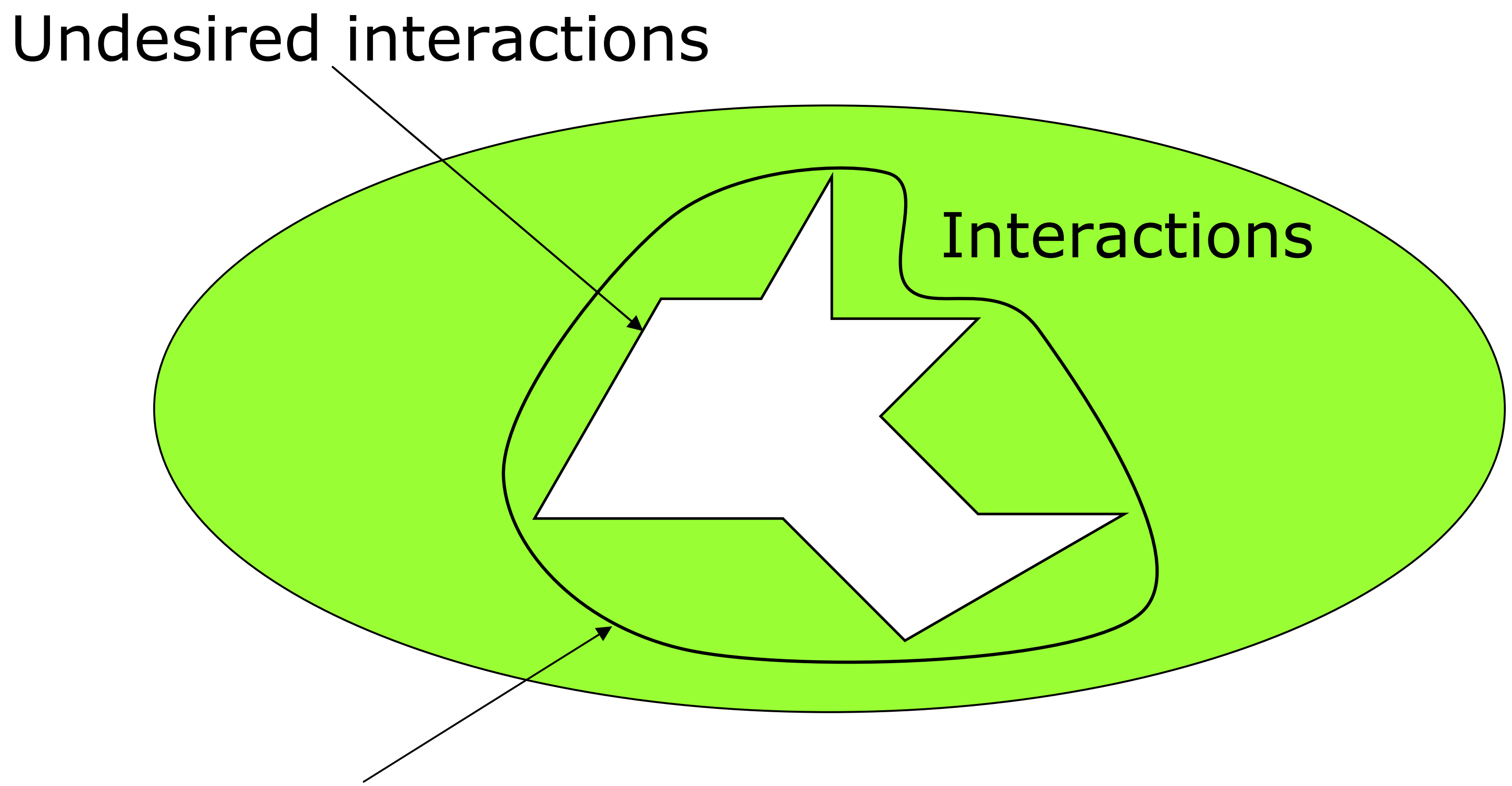
Interaction detection using properties



Interaction detection using properties



Interaction detection using properties



$\bigcup_{i}$ specific P_i = above approximation

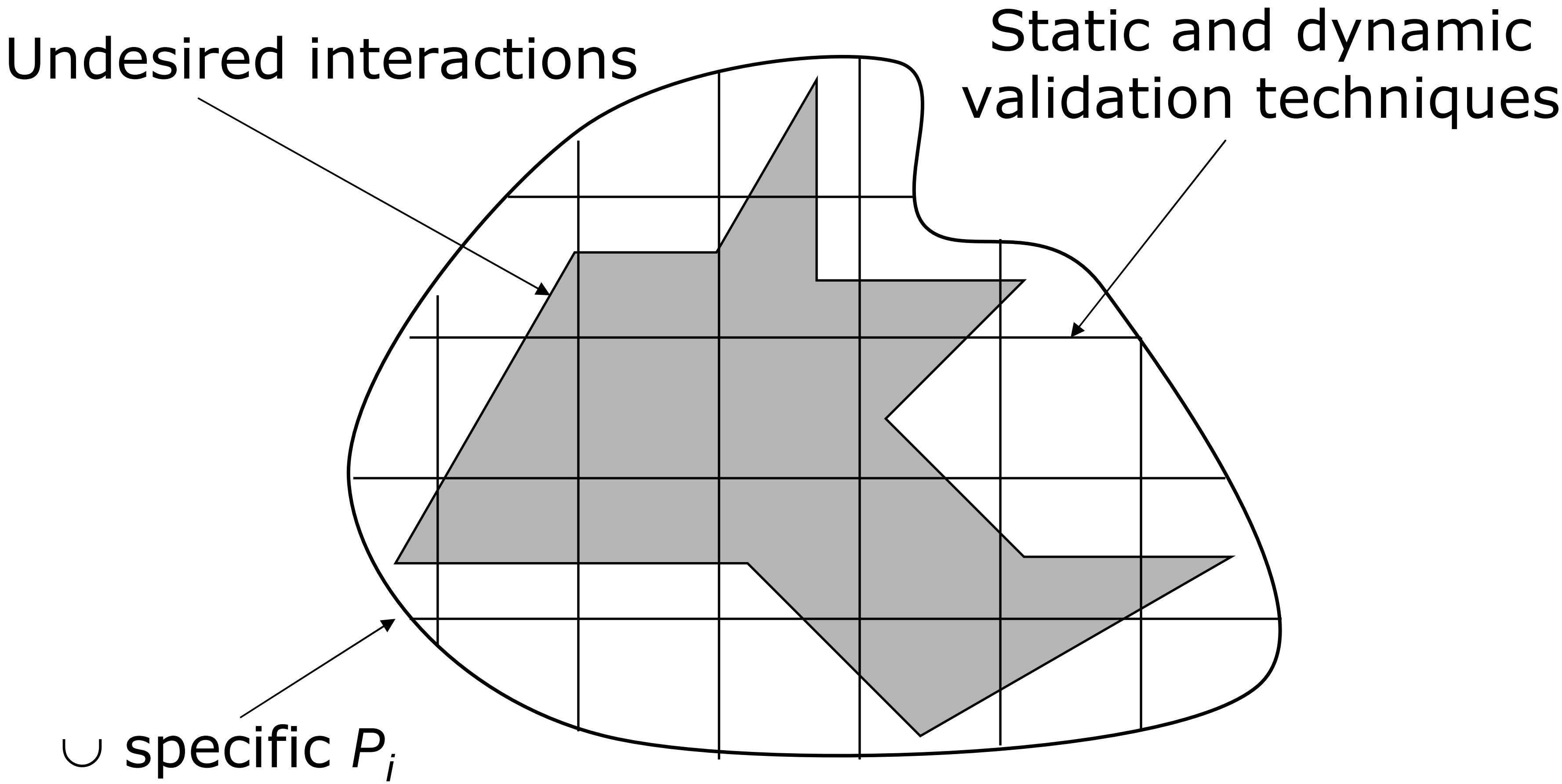
A new feature integration method with filtering

- Facts
  - Interactions : a subjective trait of service operation
  - Proof or model checking-based detection is hopeless
  - General purpose detection criteria are mostly not scalable
    - → Designers' expertise is essential
    - → Use analogy between feature integration and testing

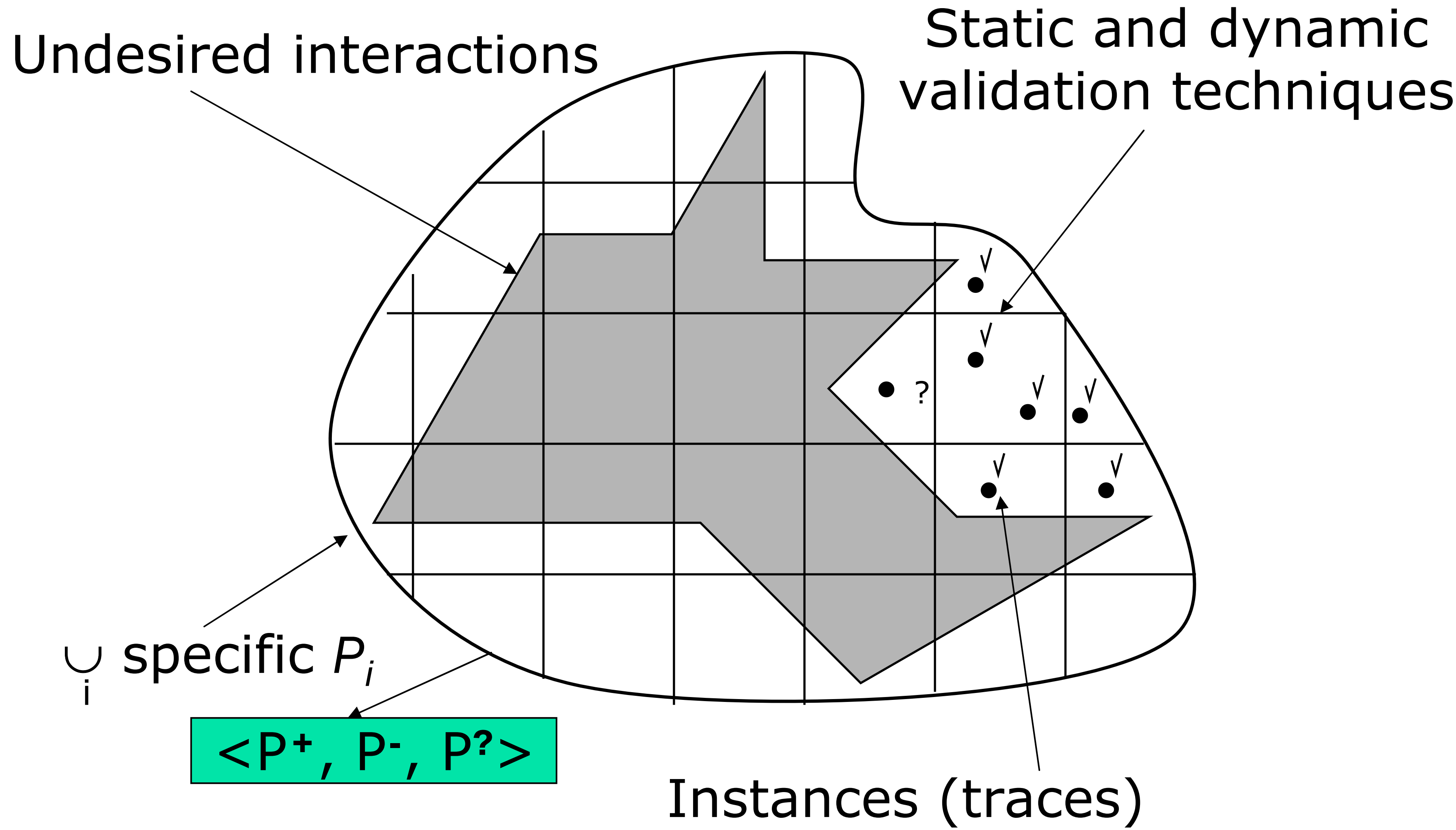
A new feature integration method with filtering

- Objectives
  - Tool-based and expert-oriented service integration methodology at the requirements level
  - Interactive specification and detection processes with automation of repetitive tasks
  - Joint and incremental elaboration of a specification <*Sys* , *Prop*> describing any service or a system resulting from the integration of several (features) services to POTS
  - Use filtering to adjust *Prop*

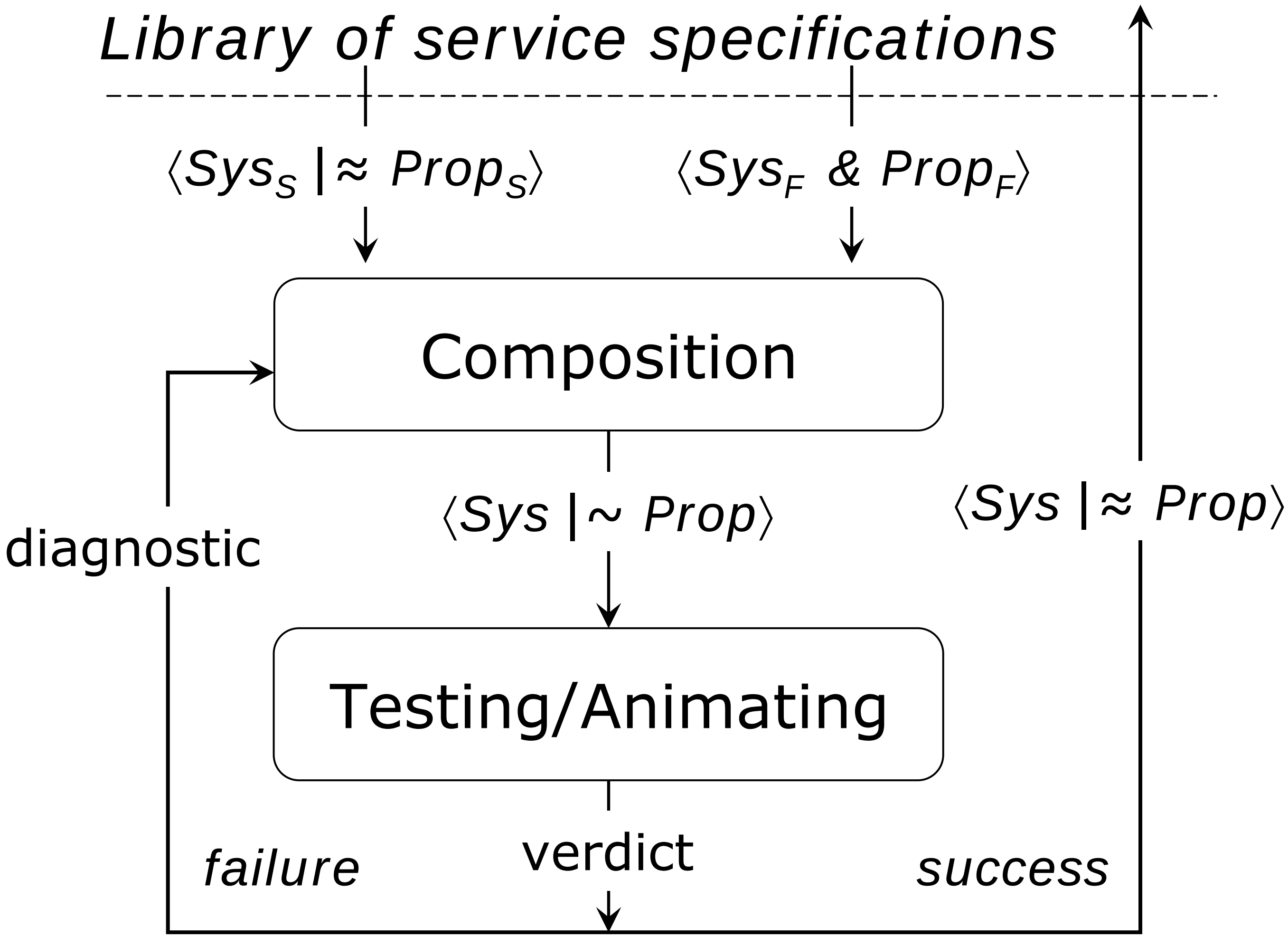
Sorting out interaction revealing properties



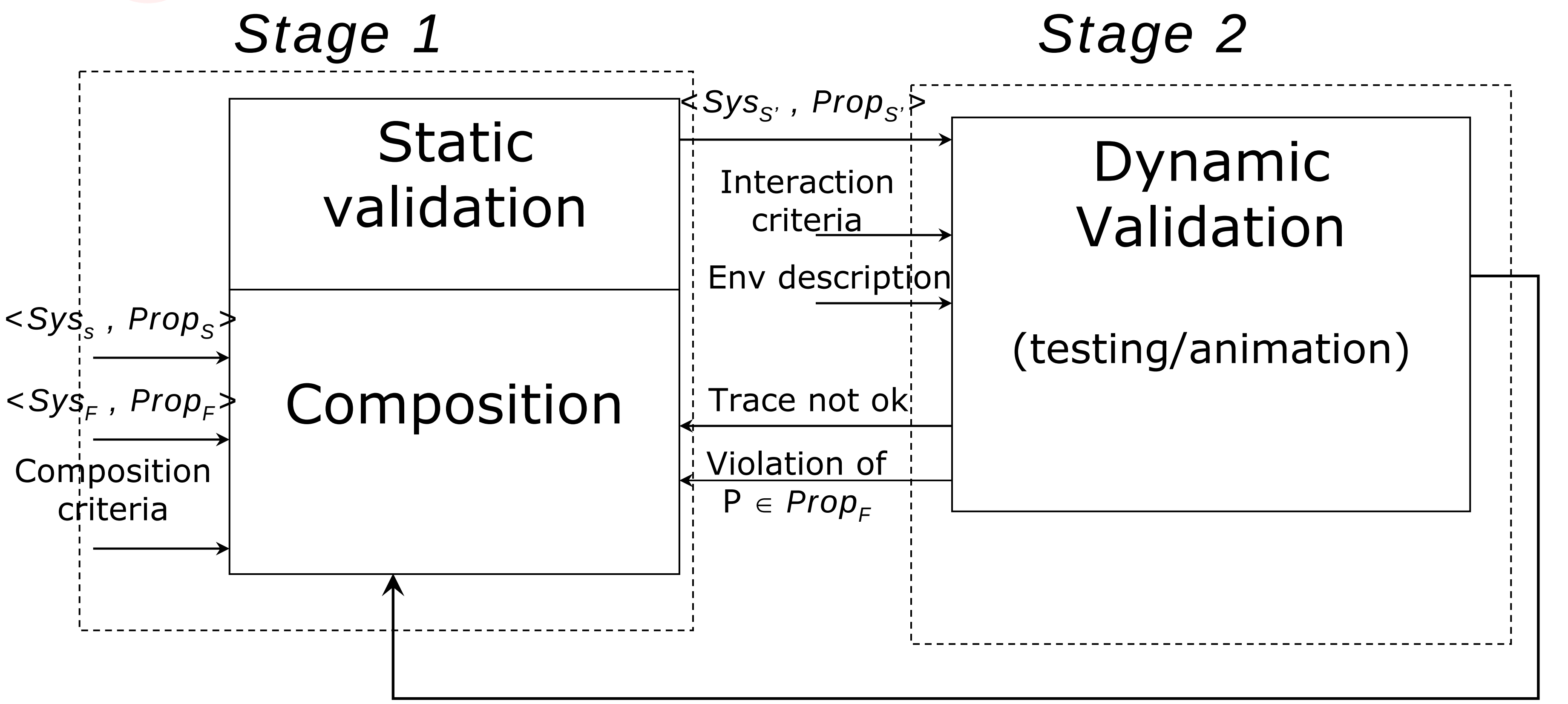
Sorting out interaction revealing properties



Feature integration principles



Feature integration process



Specification language

- No strong distinction between *Sys* and *Prop*
- Language : *State Transition Rules*
- *Sys* = rules + constraints on events
- Rules

  <{x≠y} | dialwait(x), idle(x) [dial(x,y)] calling(x,y)>

  <{x≠y} | OCS(x,y), dialwait(x), idle(x)
  [dial(x,y)] OCS(x,y), calling(x,y)>
- Constraints on events

  <{} | not idle(x) ⇒ not offhook(x)>
- Properties : invariants
  - For POTS : <{} | idle(x) ∧ not linebusy(x)>
  - For OCS : <{x≠y} | OCS(x,y) ⇒ not logcaller(x,y)>
- Formal semantics fully stated : various translations are possible

Integration : stage1

- Integrating POTS, F1, F2 : to control complexity
  - POTS + F1 , POTS + F2
  - (POTS + F1) + F2 and/or (POTS + F2) + F1
- Composition criteria : operation + consists of modifying the system or service specification on which the integration is based
- Intertwined composition and static validation steps
  - Naïve union of the specifications
  - Incremental specification adjustment :
    - Classification of *Prop* into P^+, P^-, $P^?$
    - Refinement of *Sys* : rule deletion, reinforcement
- Aid : methodology « à la B » (requirements engineering heuristics, consitency obligations) and integration historical record

Integration : stage2

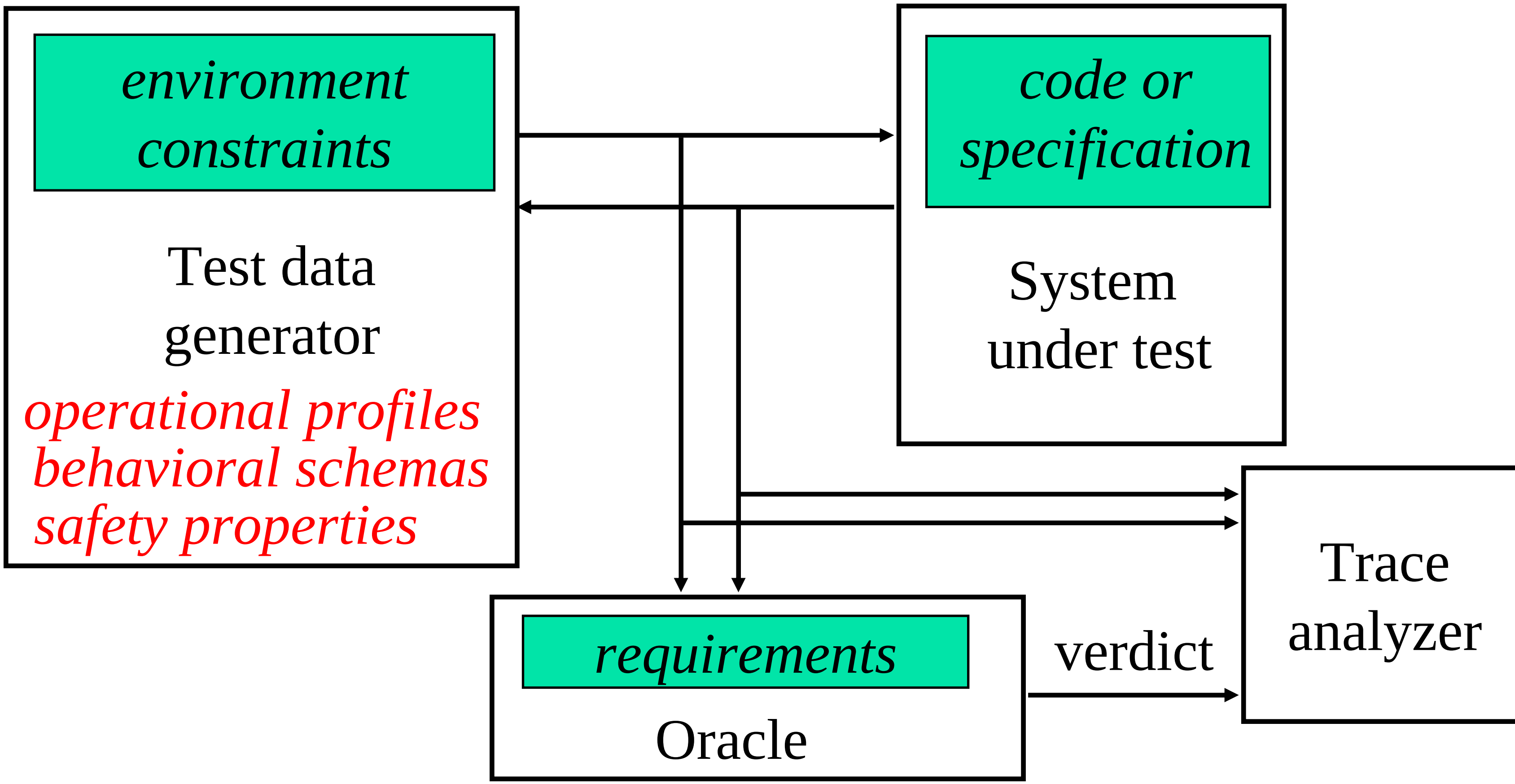
- Service animation and guided reactions from the environment

- Guides : behavioral schemas

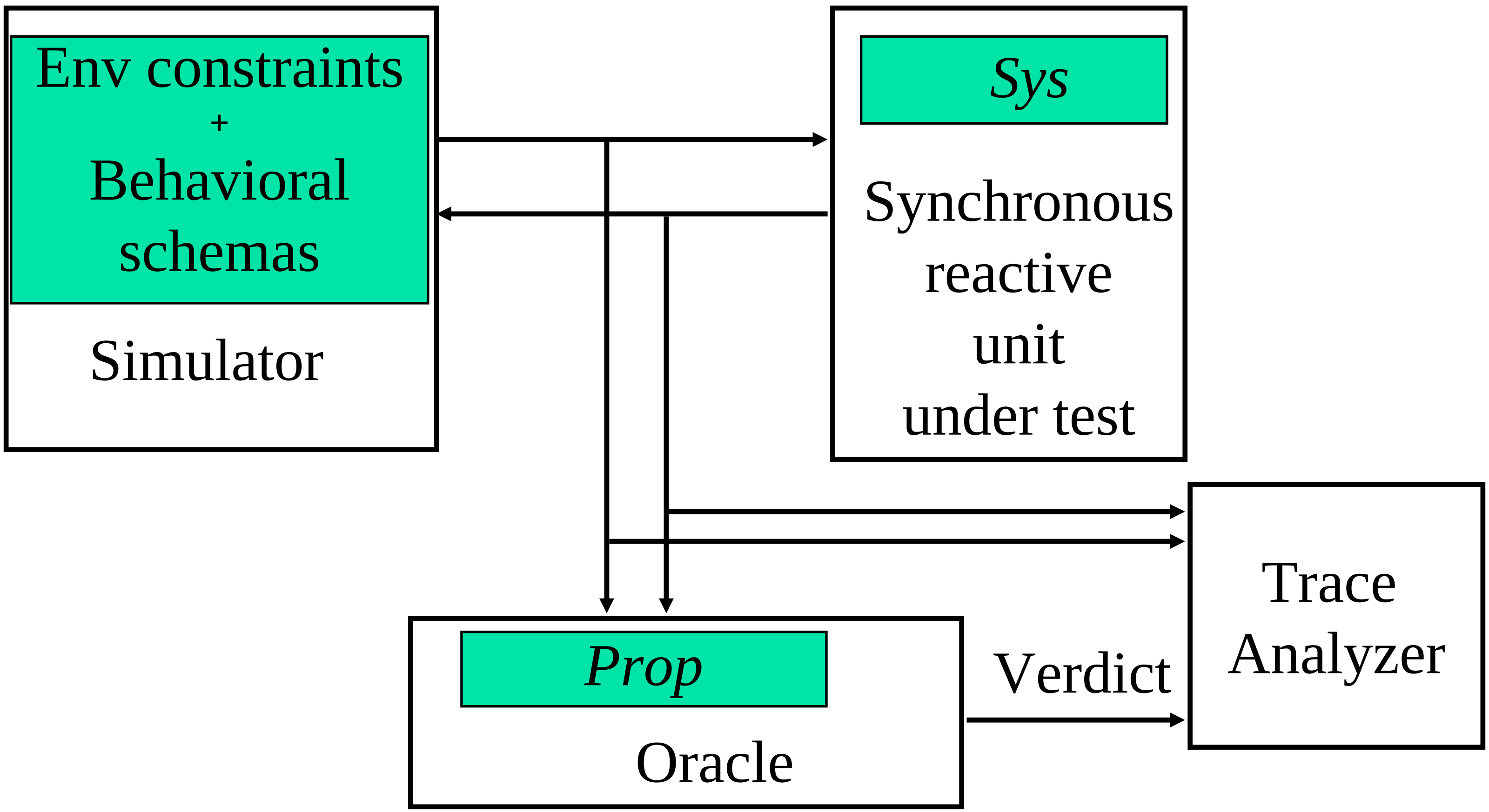
- Automatic behavioral schema generation
  - Interaction expert-based « fault model »
  - Interaction pattern language (specification language enrichment)
  - Pattern matching in a service specification

- Putting behavioral schemas into operation : Lutess

The Lutess tool



Dynamic validation using Lutess



The synchronous approach

- Instantaneous reactions to external events
- All components evolve simultaneously

THUS

- all transitions are observable
- internal actions are hidden

=> the state space is reduced

=> more concise traces

Specification synchronous animation

Pots1: < | idle(X) [offhook(X)] dialwait(X) >

+

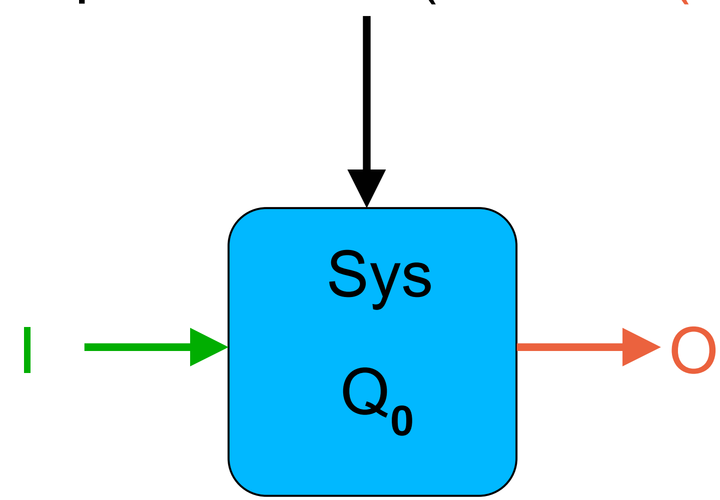
Set of values for the variables (ex: U = {A, B, C})

+

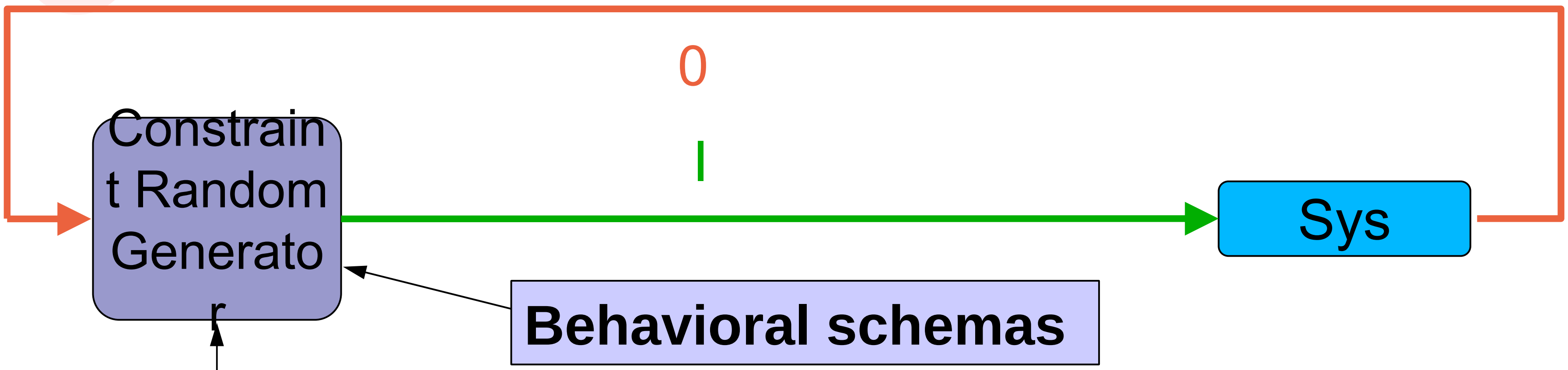
Initial state (ex : Q_0 = {idle(A), idle(B), idle(C)})

+

Service subscription parameters (ex: TCS(A, C), CFB(B, C))



Test data generation



Environment constraints :

-Instantaneous:

not dial(A, A)

-Temporal:

always_from_to(onhook(A), offhook(A), offhook(A)) $\wedge$
always_from_to(offhook(A), onhook(A), onhook(A))

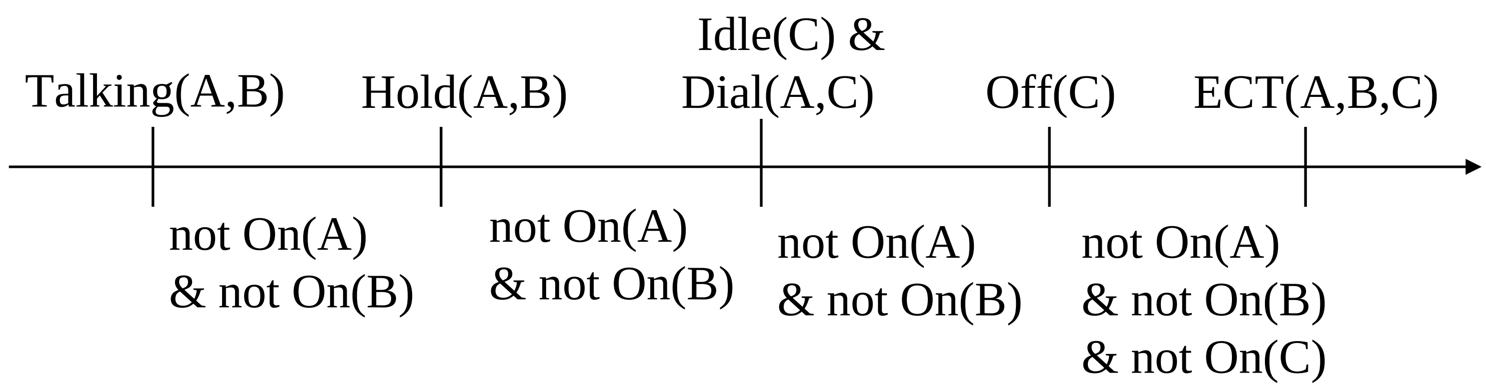
Behavioral schemas roles

- To state users' expectations (requirements)
- To guide testing in situations to be observed
- Situations of interest :
  - Suspected interactions
  - Identified by service designers' expertise
  - Related to the service bouquet model

Behavioral schema example

Guiding into a specific situation

Explicit Call Transfer service : allows a user who has two calls in progress, to connect together his two parties.



Behavioral schema construction

Using an expert-designed “fault model” which

- provides a classification of potential interactions, independent from architectures and services
- associates to every interaction-prone situation a specific interaction pattern in the form of a sequence of “normalized” actions

and applying an algorithm which automatically

- retrieves the patterns through a traversal of the state transition rule set
- generates the corresponding behavioral schemas sequences of events

Generic "fault model" for interaction classification

- Non determinism
  - Local to a single service
  - Inter services
    - One subscriber
    - Several subscribers
- Deadlock (no reaction)
- Security violation
- Bad resource handling
  - One single resource
    - The resource is persistent ……………………..
  - Two dependent resources
    - ……………………………………….
- ……………………………………………………

Interaction patterns

Controlers
Rights Level

Resources
Owner
Actions (I/R/W/T/...)
Meta actions

Examples of patterns

C.T(R) *then* C'.W(R), C ≠ C'
(one non persistent resource)

Rem_auth(C, C', A, R) ∧ RL(C) ≤ RL(C') ∧ owner(R) = C' *then* A(C', R)
(security violation)

C.I(R) *then* not C.T(R) *then* C'.I(R') ∧ R ~ R' *then* idle *then* C'.R(R
(two dependent resources with persistence)

Specification annotations

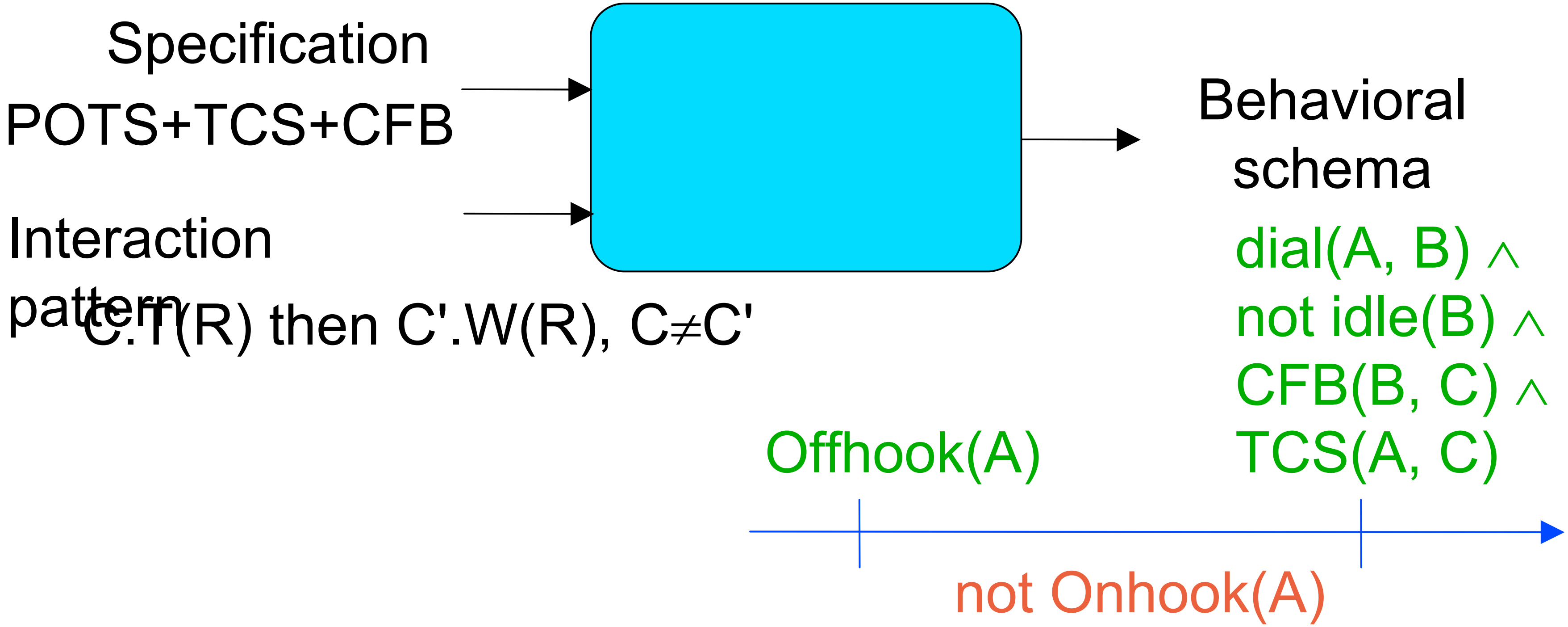
Pots2 : < A ≠ B | dialwait(A), idle(B), not TCS(A, B)

[dial(A, B)]

hearing(A, B), ringing(B, A)>

// A.I(o_callee(A), B)
// B.I(t_caller(B), A)
// B.T(t_caller(B))
// B.I(t_callee(B), B)

Behavioral schemas generation



Conclusion - Perspectives

- Our thesis :
  - Declaring an interaction undesired is subjective
  - Feature detection inefficiency comes from the huge number of potential interactions
  - Service designers' expertise is essential to classify interactions
- Tool encompassing designers' expertise is under development
- Effectiveness of the "fault model" has been confirmed by benchmarking
- Genericity of the "fault model" is being evaluated
- Efficient behavioral schemas generation is under study

Behavioral schema example

Charge Call : to charge a call to another party