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

Set of behaviors

Interactions

Undesired 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

Undesired interactions

Interactions

Strong P
Interaction detection using properties

Undesired interactions

Weak P
Interaction detection using properties

Undesired interactions

$\cup_{i} \text{ specific } P_i = \text{ above approximation}$
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

Undesired interactions

Static and dynamic validation techniques

∪ specific $P_i$
Sorting out interaction revealing properties

Undesired interactions

\[ \bigcup_{i} P_i \]

Static and dynamic validation techniques

Instances (traces)

\[ <P^+, P^-, P?> \]
Feature integration principles

Library of service specifications

\[ \langle Sys_S | \approx Prop_S \rangle \]
\[ \langle Sys_F \& Prop_F \rangle \]

Composition

\[ \langle Sys | \sim Prop \rangle \]

Testing/Animating

failure

verdict

success
Feature integration process

**Stage 1**

Static validation

Composition

Composition criteria

**Stage 2**

Dynamic Validation

(testing/animation)

Violation of $P \in Prop_F$

Trace not ok

Env description

Interaction criteria

$<Sys_{s'}, Prop_{s'}>$

$<Sys_{s}, Prop_{s}>$

$<Sys_{F}, Prop_{F}>$
Specification language

- No strong distinction between Sys and Prop
- Language: State Transition Rules
- Sys = rules + constraints on events
- Rules
  - \(\{x\neq y\} | \text{dialwait}(x), \text{idle}(x) \ [\text{dial}(x,y)] \ \text{calling}(x,y)\rangle\)
  - \(\{x\neq y\} | \text{OCS}(x,y), \text{dialwait}(x), \text{idle}(x)\)
    - \[\text{dial}(x,y)] \ \text{OCS}(x,y), \text{calling}(x,y)\rangle\)
- Constraints on events
  - \(\{\} | \text{not idle}(x) \Rightarrow \text{not offhook}(x)\rangle\)
- Properties: invariants
  - For POTS: \(\{\} | \text{idle}(x) \land \text{not linebusy}(x)\rangle\)
  - For OCS: \(\{x\neq y\} | \text{OCS}(x,y) \Rightarrow \text{not logcaller}(x,y)\rangle\)
- Formal semantics fully stated: various translations are possible
Integration : stage 1

- 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

- **Environment constraints**
- **Test data generator**
  - operational profiles
  - behavioral schemas
  - safety properties
- **Code or specification**
- **System under test**
- **Requirements**
- **Oracle**
- **Verdict analyzer**
Dynamic validation using Lutess

- Env constraints + Behavioral schemas
- Simulator
- Prop
- Oracle
- Sys
- Synchronous reactive unit under test
- Trace Analyzer
- Verdict

Oracle

Sys

Synchronous reactive unit under test
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)) ∧
  - 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
Guiding into a specific situation

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

Talking(A,B)  Hold(A,B)  Idle(C) & Dial(A,C)  Off(C)  ECT(A,B,C)

not On(A) & not On(B)  not On(A) & not On(B)  not On(A) & not On(B)  not On(A) & not On(B)  not On(A) & not On(B) & not On(C)
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

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 \neq B \mid \text{dialwait}(A), \text{idle}(B), \text{not TCS}(A, B) \\
[dial(A, B)] \\
\text{hearing}(A, B), \text{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

Specification
POTS+TCS+CFB

Interaction pattern
C.T(R) then C'.W(R), C≠C'

Behavioral schema

dial(A, B) \land
not idle(B) \land
CFB(B, C) \land
TCS(A, C)

Offhook(A)

not Onhook(A)
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

<table>
<thead>
<tr>
<th>pre DialTone(A) and Dial(A,0)</th>
<th>Dial(A,B)</th>
<th>Dial(A,C)</th>
<th>Dial(A,code(C))</th>
</tr>
</thead>
<tbody>
<tr>
<td>not On(A)</td>
<td>not On(A)</td>
<td>not On(A)</td>
<td></td>
</tr>
</tbody>
</table>