Compatibility Flooding: Measuring Interaction of Services Interfaces

Meriem OUEDERNI[1] Uli FAHRENBERG[2]
Axel LEGAY[3] Gwen SALAÜN[4]

[1]IRIT / Toulouse INP, Toulouse, France

[2]École polytechnique, Palaiseau, France

[3]INRIA, Rennes, France

[4]INP, INRIA, LIG, Grenoble, France

The 32rd Symposium On Applied Computing
Morocco 2017

Compatibility Flooding: Measuring Interaction of Services Interfaces

Meriem OUEDERNI[1] Uli FAHRENBERG[2]
Axel LEGAY[3] Gwen SALAÜN[4]

[1]IRIT / Toulouse INP, Toulouse, France

[2]École polytechnique, Palaiseau, France

[3]INRIA, Rennes, France

[4]INP, INRIA, LIG, Grenoble, France

The 32rd Symposium On Applied Computing
Morocco 2017

Introduction (1/3)

- Service-based systems are built using existing software applications called services
- Services are loosely-coupled, independently developed, and accessed through their public interfaces, *i.e.*, signature and interaction protocols
- Service interfaces are often incompatible, *e.g.*, missing message, missing parameter, deadlock, etc.
- Interface compatibility must be checked in order to avoid erroneous behaviours and ensure the safe reuse of services

Introduction (2/3)

Different compatibility notions exist [Yellin&Strom],[Brand&Zafiropulo], [de Alfaro&Henzinger], [Bordeaux&al], etc., *e.g.*, two services are compatible if:

- They can at least engage one communication sequence until reaching a global final state: One-Path
- Their interaction does not deadlock: Deadlock-Freeness
- All reachable request (emission) must be replied (received): Unspecified-Receptions
- They have opposite behaviours: Opposite-Behaviours
- . . .

Introduction (3/3)

Limitations of existing approaches are:

- They commonly return a Boolean (True/False) result. However, services are often incompatible, and
  - A False result does not differentiate between slightly and totally incompatible services
  - A False result gives no information which parts of service protocols are compatible or not
- A very few recent approaches compute a numeric compatibility measure. But,
  - The checked interfaces do not consider value-passing and internal τ actions
  - The measuring process consists in simple (not iterative) protocol traversal
  - A unique compatibility notion is considered

Our Proposal

- We consider value-passing and τ actions in the interface description model and the verification process
- We propose a generic Framework to automatically measure the interface compatibility: a numerical result is returned
- The framework is parameterised with different notions organised into bidirectional and unidirectional classes. This talk presents a bidirectional notion: unspecified receptions
- We consider two-step measuring process:
    - Computation of static compatibility
    - Computation of (behavioural) protocol compatibility using the static compatibility: iterative protocol traversal and flooding techniques are considered
- The measuring process also returns the mismatch list, and it is fully automated into our Comparator prototype tool

Outline

Symbolic Transition System (STS)

Definition

STS: (A: Alphabet, S: States, I: Initial state, F: Final states, T: Transitions) where:

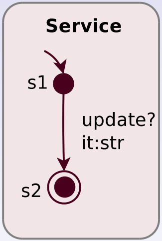
- A: (M: message, D: direction (! or ?) ,PL: typed parameter list), or τ action
- $I \in S$, $F \subseteq S$, $T \subseteq S \setminus F \times A \times S$

- The STS is very convenient for formal description and verification of service behaviours
- The STS's operational semantics is synchronous
- This model can be easily be derived from existing platform languages, *e.g.*, WF, BPEL for Web services

Symbolic Transition System (STS)

Definition

STS: (*A*: Alphabet, *S*: States, *I*: Initial state, *F*: Final states, *T*: Transitions)

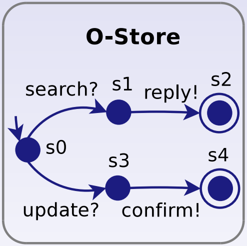


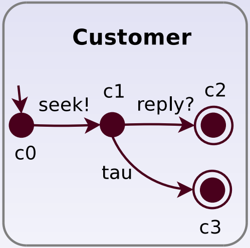
Outline

Unspecified Receptions

Definition

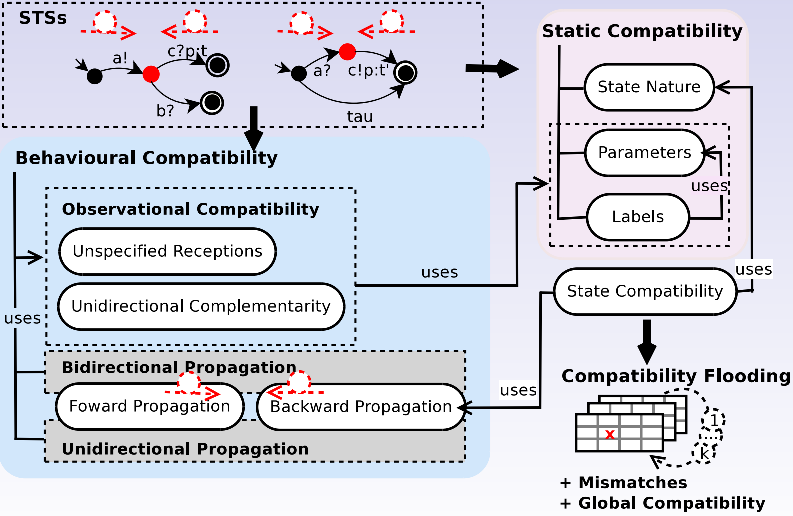
Each service must receive all messages emitted by its partner at all reachable states, and both services must be free of deadlocks





Outline

Overview of Our Approach



Measures

State Nature

- Given a global state $(s_1, s_2) \in S_1 \times S_2$, state nature compatibility is equal to 1 if both states have the same nature, *i.e.*, initial, final, or none of them. Otherwise, this measure is 0.

Parameters

- This measure is computed from the comparison of emitted and received parameter types, order, and number.

Labels

- Two labels are totally incompatible, $\mathit{lab\text{-}comp}(l_1, l_2) = 0$, if they have the same direction.
- Otherwise, $\mathit{lab\text{-}comp}(l_1, l_2)$ compares the semantic distance between label names (using Wordnet), and the parameter types.

Preliminaries

Being given two STSs, $STS_{i \in \{1,2\}} = (A_i, S_i, I_i, F_i, T_i)$:

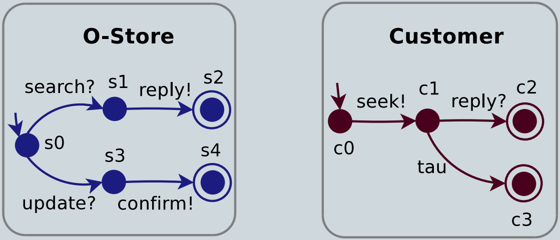
- Definition: two states are compatible if their backward and forward neighbouring states are compatible
- Measuring techniques: a compatibility flooding algorithm and an iterative computation
- Result: a matrix $COMP^k_{CN,D}$, a list of mismatches, and a global compatibility measure:
  - $COMP^k_{CN,D}[s_1, s_2]$ is the compatibility measure of the global state (s_1, s_2) at the k^{th} iteration
  - CN is the compatibility notion and $D \in \{\leftrightarrow, \rightarrow\}$
  - $\forall (s_1, s_2) \in S_1 \times S_2,\ COMP^0_{CN,D}[s_1, s_2] = 1$
  - $COMP^k_{CN,D}[s_1, s_2] =$ **?**

Observational Compatibility

The observational compatibility is returned by the function $obs\text{-}comp^k_{UR,\leftrightarrow}((s_1, s_2))$:

1. It returns 0 if there is a deadlock
2. It returns 1 if every emission in s_1 (s_2, resp.) perfectly matches a reception in s_2 (s_1, resp.), and both protocols evolve into compatible states
3. Otherwise, it is computed from the best compatibility obtained from the comparison of every emission in s_1 (s_2, resp.) and the receptions in s_2 (s_1 resp.) leading to the best neighbours
   - The best compatibility at the k^{th} iteration is determined by the maximal value of $lab\text{-}comp(l_1, l_2) * COMP^k_{CN,D}[s'_1, s'_2]$

Observational Compatibility



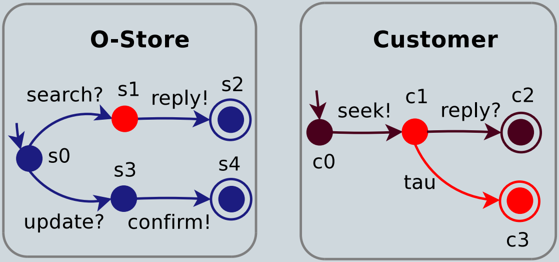
$$obs\text{-}comp^{1}_{UR,\leftrightarrow}((s_0, c_0))$$
$$= lab\text{-}comp(seek!, search?) *$$
$$COMP^{0}_{UR,\leftrightarrow}[s_1, c_1]$$
$$= 1$$

State Compatibility

This measure, $\textit{state-comp}^{k}_{UR,\leftrightarrow}((s_1, s_2))$, is computed from:

- The bidirectional propagation of the compatibility measure returned for the neighbouring states, *i.e.*, consideration of both service point of view:
  - The existence of τ transitions requires to compute the compatibility on the target states
  - The observable transitions are compared using $\textit{obs-comp}^{k}_{UR,\leftrightarrow}$
- The state nature compatibility

State Compatibility



$$fw\text{-}propag^{1}_{UR,\leftrightarrow}((s_1, c_1)) = \frac{1}{2} * \left[\frac{fw\text{-}propag^{1}_{UR,\leftrightarrow}((s_1,c_3)) + obs\text{-}comp^{1}_{UR,\leftrightarrow}((s_1,c_1))}{2} + obs\text{-}comp^{1}_{UR,\leftrightarrow}((s_1, c_1))\right]$$

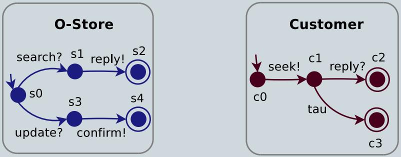
- $fw\text{-}propag^{1}_{UR,\leftrightarrow}((s_1, c_3)) = obs\text{-}comp^{1}_{UR,\leftrightarrow}((s_1, c_3)) = 0$
- $obs\text{-}comp^{1}_{UR,\leftrightarrow}((s_1, c_1))$
  $= lab\text{-}comp(reply?, reply!) * COMP^{0}_{UR,\leftrightarrow}[s_2, c_2] = 1$

Compatibility Flooding

Last Measuring Step

$COMP^{k}_{CN,D}[s_1, s_2]$ is computed from its previous value $COMP^{k-1}_{CN,D}[s_1, s_2]$ and *state-comp*$^{k}_{CN,D}((s_1, s_2))$.

Matrix $COMP^{7}_{UR,\leftrightarrow}$

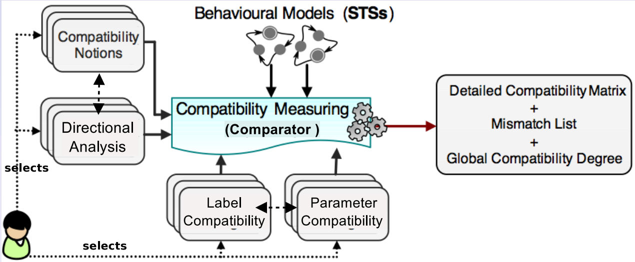


	c_0	c_1	c_2	c_3
s_0	0.95	0.17	0.01	0.01
s_1	0.01	0.82	0.01	0.32
s_2	0.01	0.26	0.95	0.51
s_3	0.01	0.47	0.01	0.16
s_4	0.01	0.26	0.75	0.51

Characteristics of our Compatibility Measure

- The compatibility flooding ensures that the effect of any detected mismatch must be propagated until the initial states
- Two protocols are compatible iff $COMP^k_{CN,D}[I_1, I_2] = 1$
- Incompatible protocols can be compared using a global compatibility degree computed from $COMP^k_{CN,D}$
- The global compatibility degree helps for:
  - Ranking and selecting services from a huge number of candidates
  - Simplifying further processing to solve interface mismatches

Comparator



- The Comparator implementation is generic, modular, extensible, and automated
- Experiments have been applied on many examples (> 115, some of them consist of hundreds of states and transitions)
- Some real-world examples are available at `www.lcc.uma.es/~meriem/comparator.html`

Proof of Convergence

- $COMP^k_{CN,D}$ is a function of $COMP^{k-1}_{CN,D}$ and *state-comp*$^k_{CN,D}$:

$$COMP^k_{CN,D} = F(COMP^{k-1}_{CN,D}) := \frac{COMP^{k-1}_{CN,D} + \textit{state-comp}^k_{CN,D}}{2}$$

  - these are (square) matrices; *state-comp*$^k_{CN,D}$ can be expressed as a (complicated!) function of $COMP^{k-1}_{CN,D}$
- Need to show that the iteration $COMP^{k-1}_{CN,D} \mapsto COMP^k_{CN,D} = F(COMP^{k-1}_{CN,D})$ converges
- Proof: Find $\lambda < 1$ so that for all matrices M_1, M_2, $\|F(M_1) - F(M_2)\| \leq \lambda \|M_1 - M_2\|$
  - $\|M\| = \max_{i,j} M[i,j]$ is supremum metric
- Hence F is λ-Lipschitz continuous, so for every $\epsilon > 0$ there is K such that for all $k \geq K$, $\|COMP^k_{CN,D} - COMP^{k-1}_{CN,D}\| < \epsilon$
  - Banach fixed-point theorem

Outline

1. Model of Services
2. Interface Compatibility
   - Bidirectional Notion
3. Compatibility Measuring
   - Static Compatibility
   - Behavioural Compatibility: Forward Computation
   - Prototype Tool
   - Proof of Convergence
4. **Concluding Remarks**

Main Contributions & Perspectives of Our Work

- Generic and extensible framework for measuring the compatibility of service interfaces considering different notions
- Numerical measure of service compatibility: Boolean compatibility can be also detected
- Tool support and application to semi-automated service adaptation (ACIDE and DINAPTER tools)

- Automatic generation of adaptation contract
- Automatic management of service evolution
- Checking the service protocol compatibility under the asynchronous communication semantics

Main Contributions & Perspectives of Our Work

- Generic and extensible framework for measuring the compatibility of service interfaces considering different notions
- Numerical measure of service compatibility: Boolean compatibility can be also detected
- Tool support and application to semi-automated service adaptation (ACIDE and DINAPTER tools)

- Automatic generation of adaptation contract
- Automatic management of service evolution
- Checking the service protocol compatibility under the asynchronous communication semantics

THANK YOU