Supporting Composition *of* Distributed Business Processes

astro

Analysis of Communication Models in Web Service Compositions

Marco Pistore
pistore@dit.unitn.it
University of Trento

Joint work with **Raman Kazhamiakin** and **Luca Santuari**

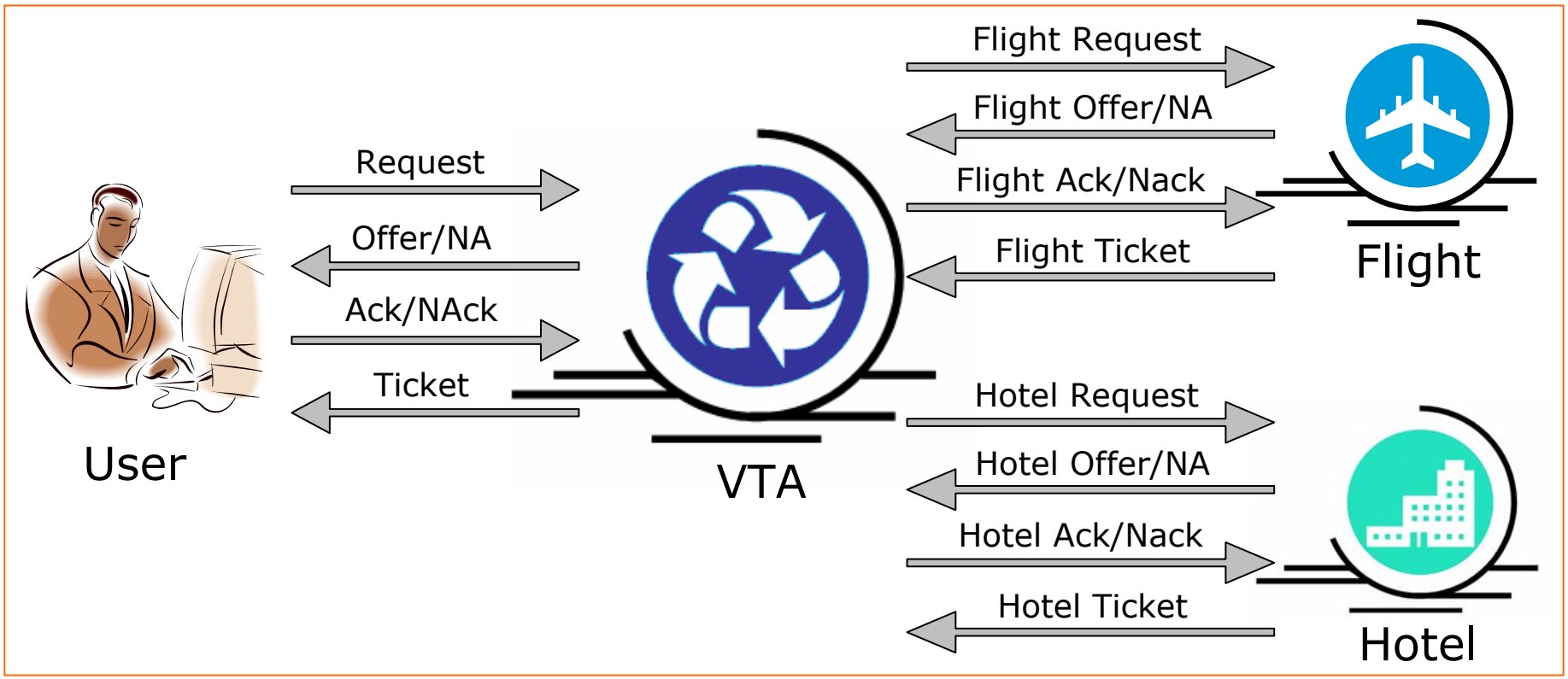
Context

- **ASTRO project**
  - Provide tools that support the design and execution of **service compositions**
  - Component services are stateful and long-running (e.g., **WS-BPEL** services)
- One of the techniques provided by ASTRO is **formal verification**
  - Formal check of the **correctness of the composition behavior**
- **Interaction** mechanisms are complex and implementation-dependent
  - Complex queues and message management mechanisms
  - Scenarios with message overpasses and losses are possible
  - Diversity of implementations
- Necessity to define an appropriate formal **communication models**
  - Tradeoff between expressiveness and complexity of the analysis

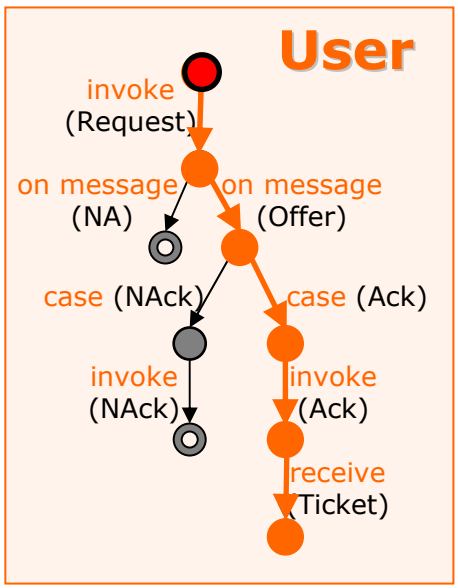
Motivating example

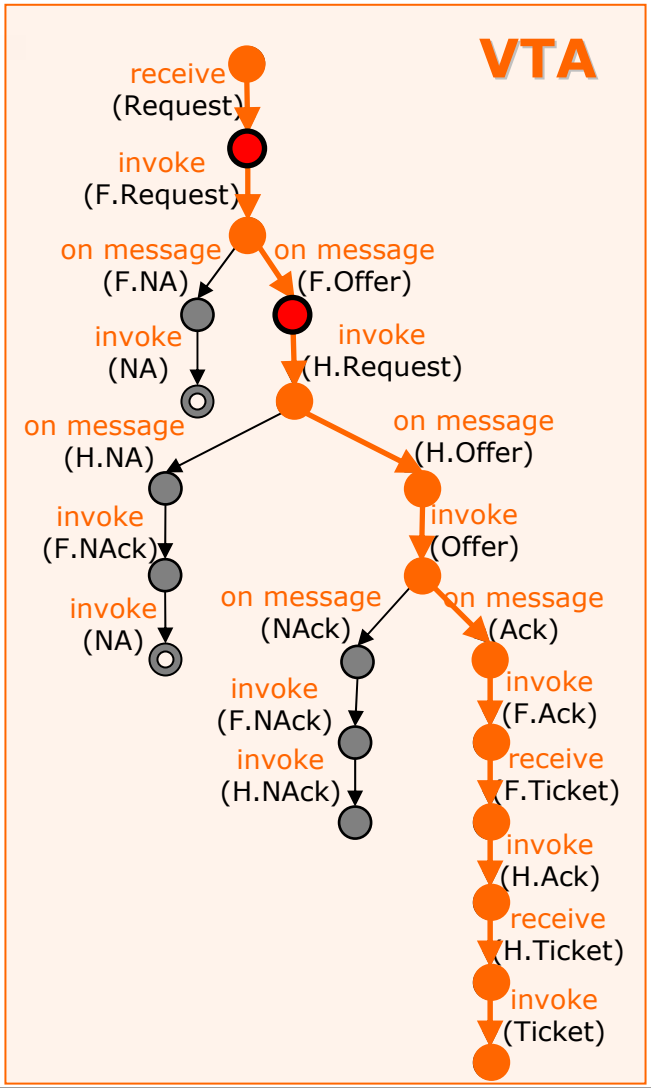
Motivating example: Virtual Travel Agency

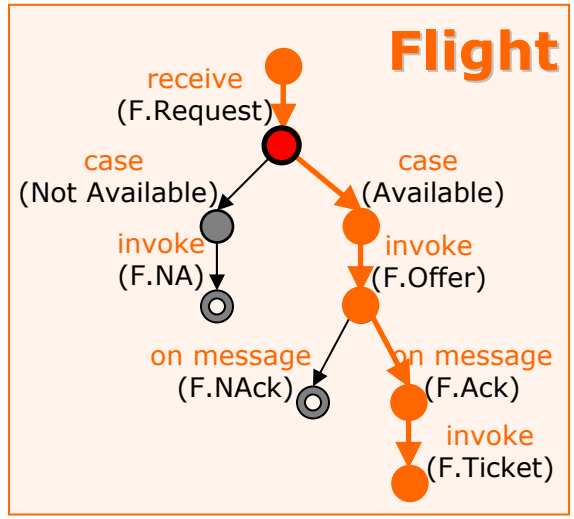
- Provide combined flight and hotel booking service
- Integrate separate **Hotel** and **Flight** booking services
- Participants are represented with their **BPEL** specifications

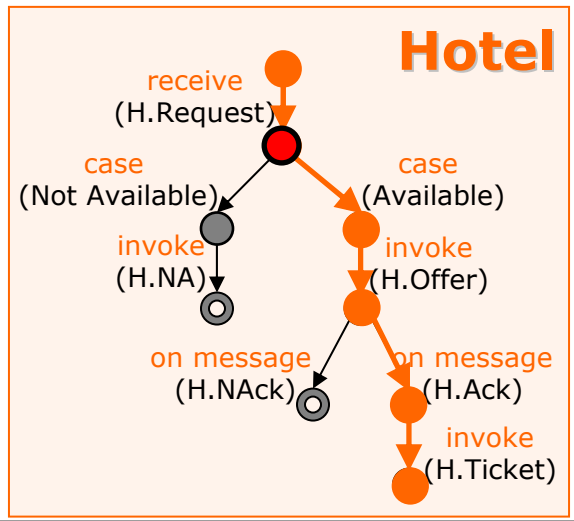


VTA Processes





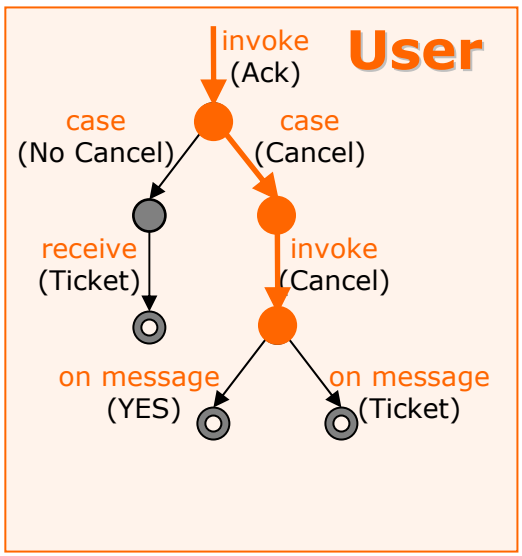


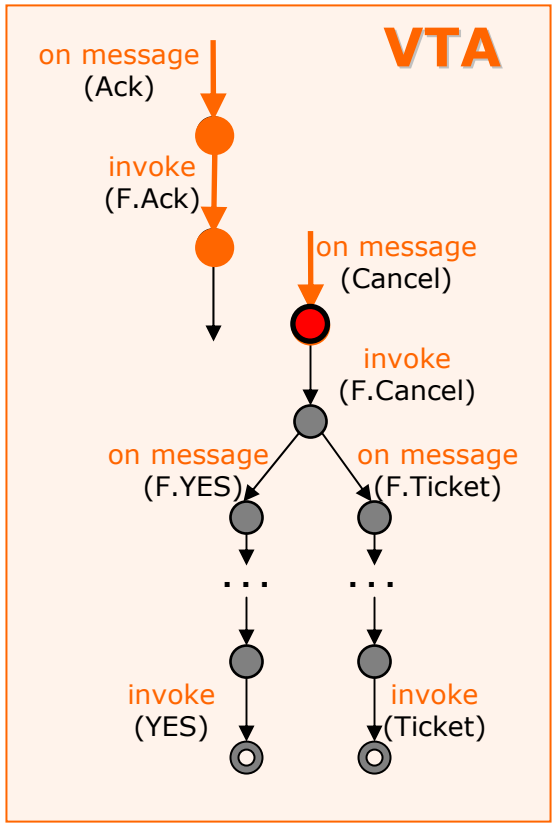


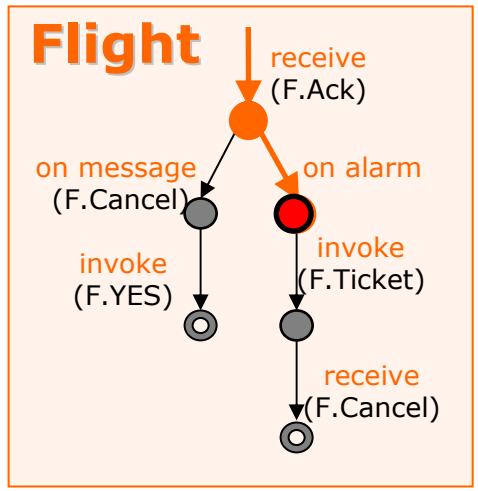
Composition Properties

- The composition is **synchronizable**
  - At any moment of time only one component emits a message
  - The receiver is immediately ready to consume the message
- **Synchronous communication model**
  - Components synchronize on shared actions
  - Efficient reasoning techniques
  - Universally used in verification tools for web service compositions
- The synchronous communication model is **adequate** for synchronizable compositions
  - The presence of queues in the implementation does not add new behaviors
- Not applicable to the wide range of systems
  - E.g. cancellation scenarios

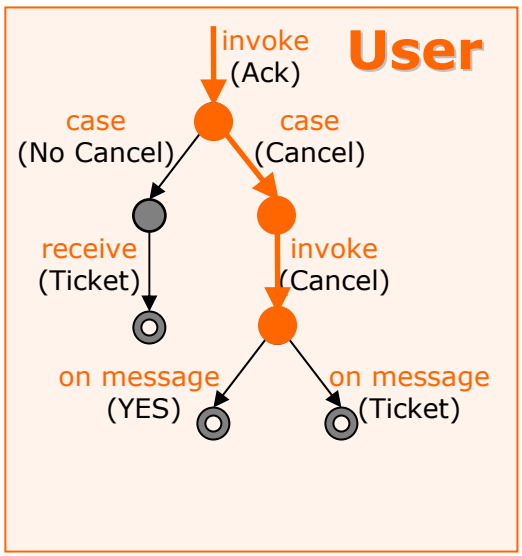
VTA Processes – Cancellation

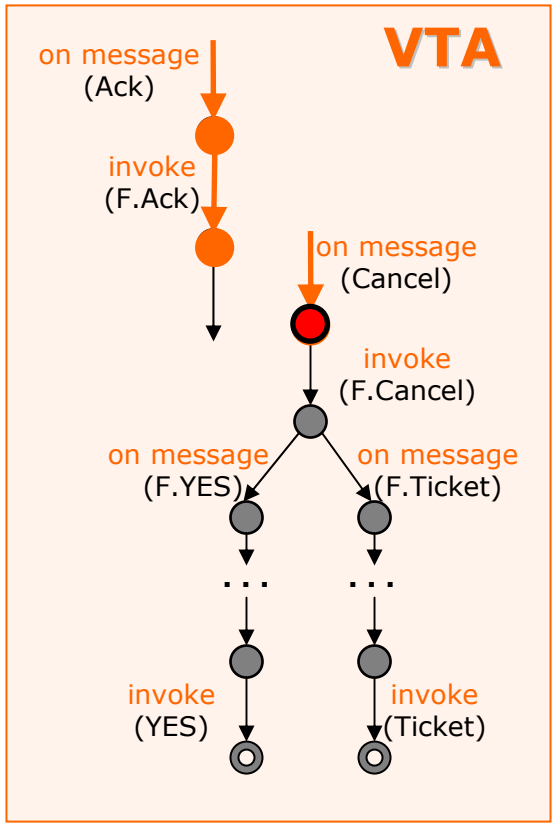


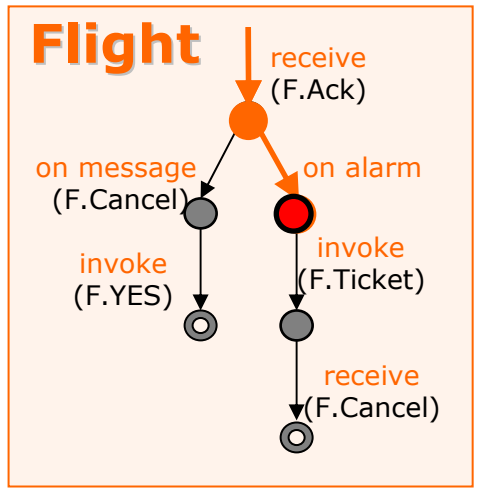




VTA Processes – Cancellation

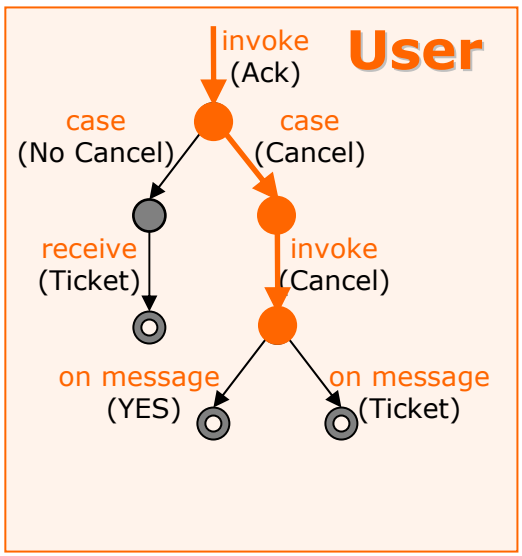


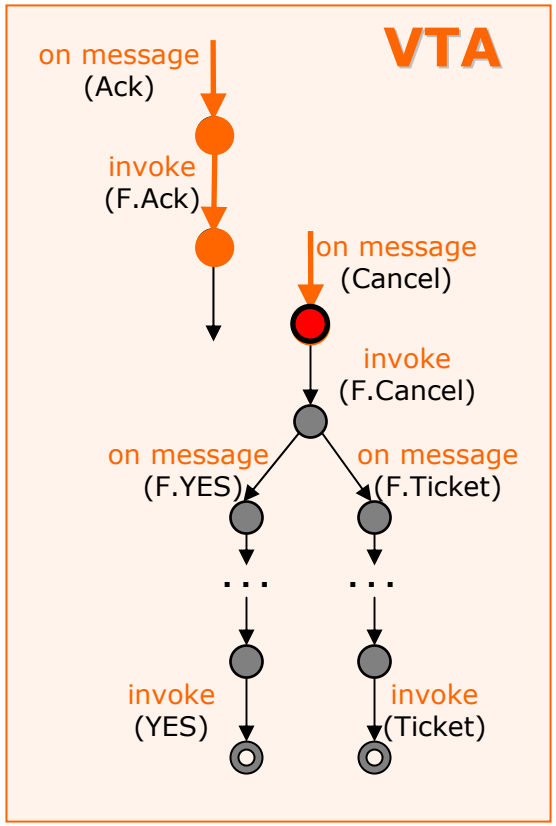


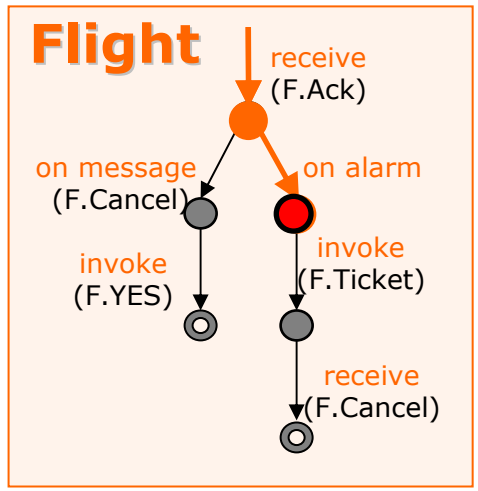


Deadlock?

VTA Processes – Cancellation



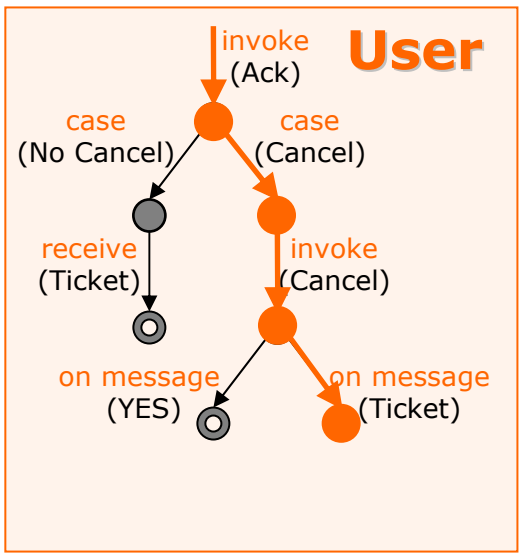


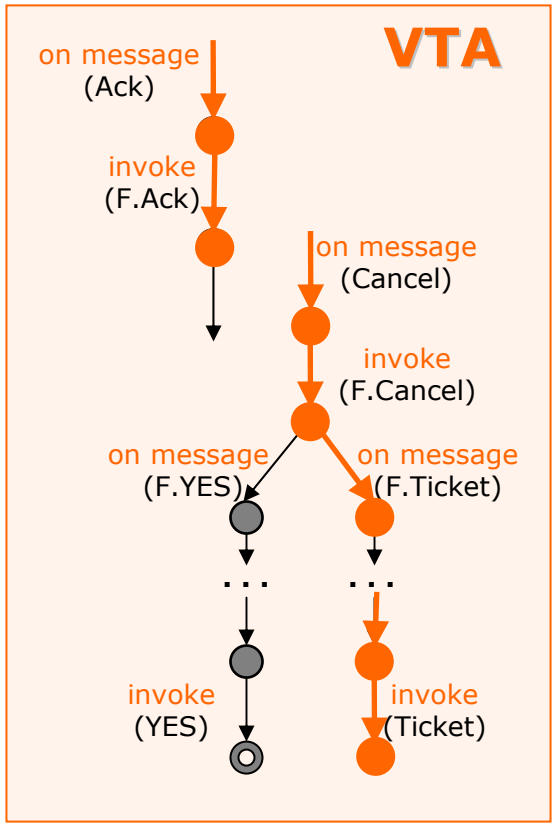


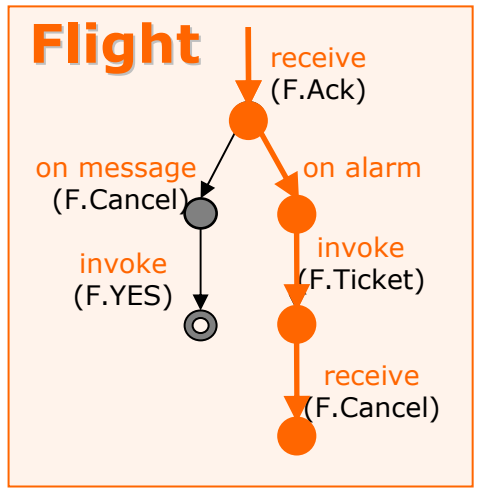
Deadlock?

No!

VTA Processes – Cancellation



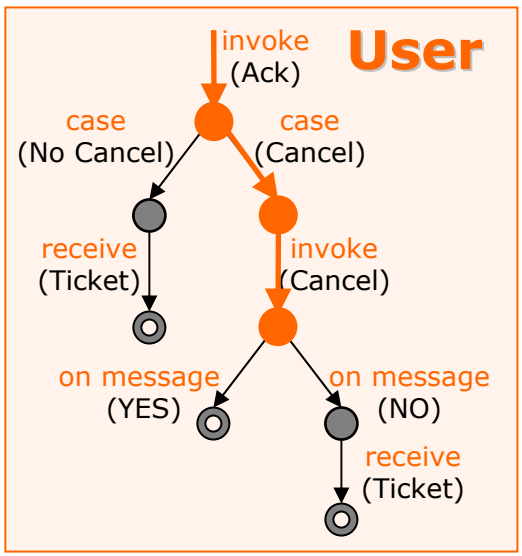


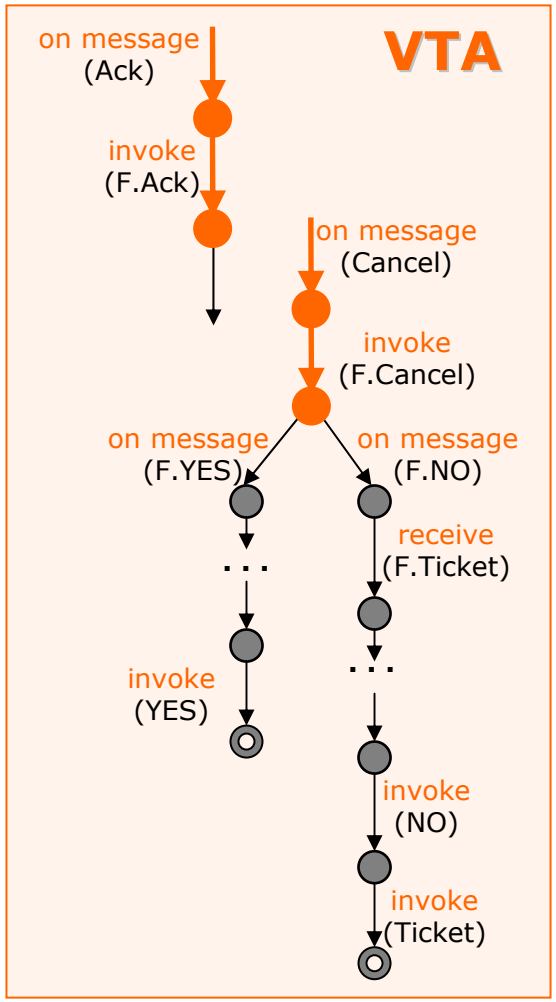


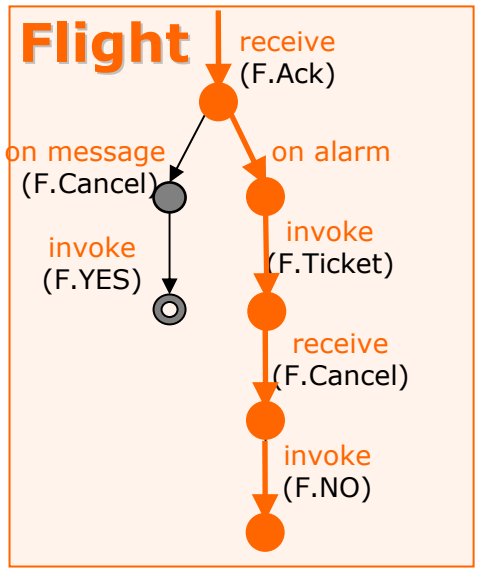
Composition Properties

- The synchronous communication model is violated
  - The concurrent emission of messages is possible in the same time point: *F.Cancel* and *F.Ticket*
- The real execution is correct
  - Engines support non-blocking message emissions and message queues
  - In the scenario, all message are eventually consumed, and cancellation is performed correctly
- An **adequate communication model** is needed to formally verify this scenario!
- **Asynchronous ordered** communication model
  - Emission and reception of a message do not have to happen at the same time
  - The order of message emission and the order of message reception between two processes should be the same (no message overpass)
  - Intuitively: an ordered queue between each pair of processes

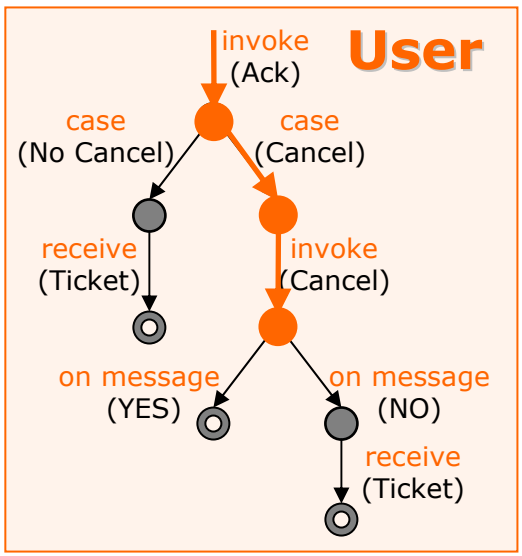
VTA Processes – Complex Cancellation

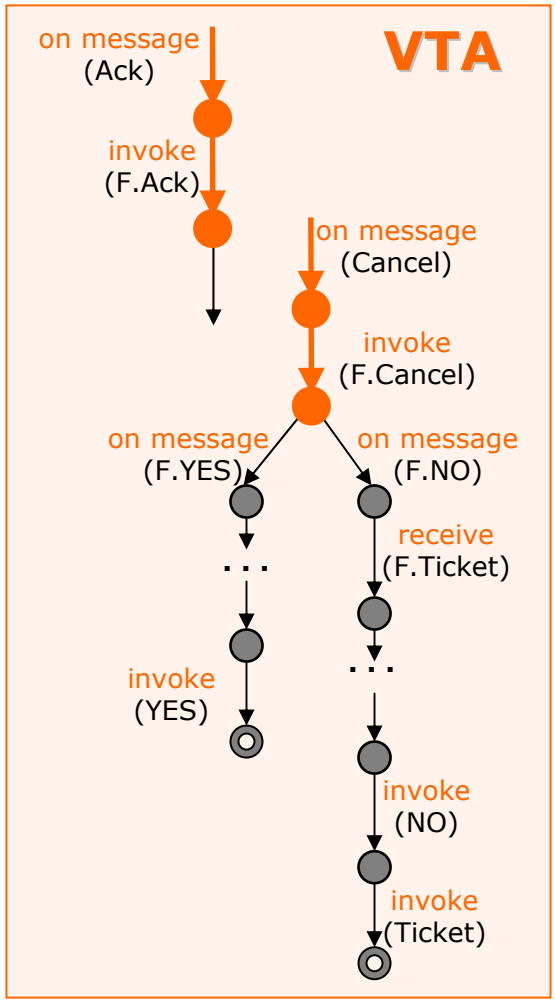


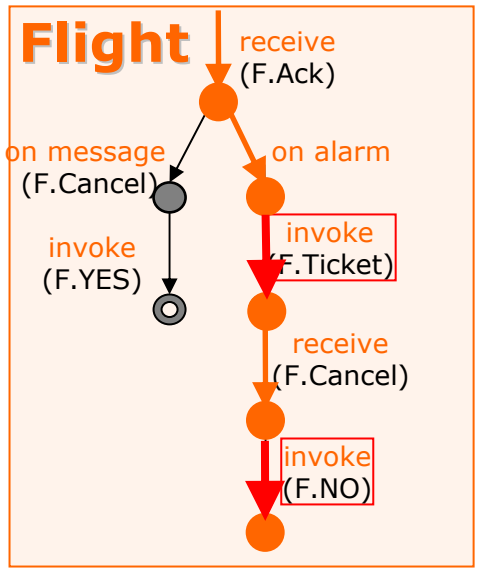




VTA Processes – Complex Cancellation



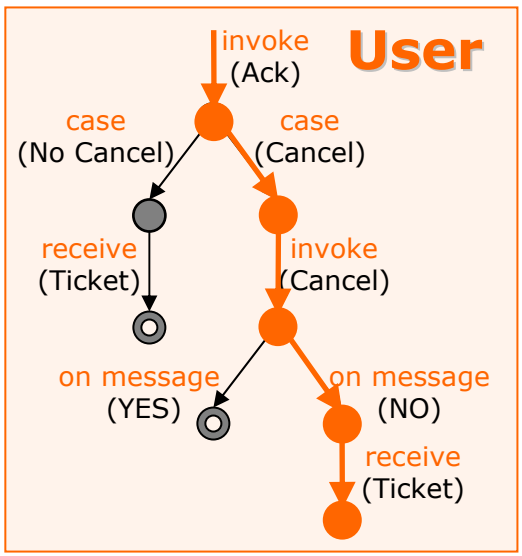


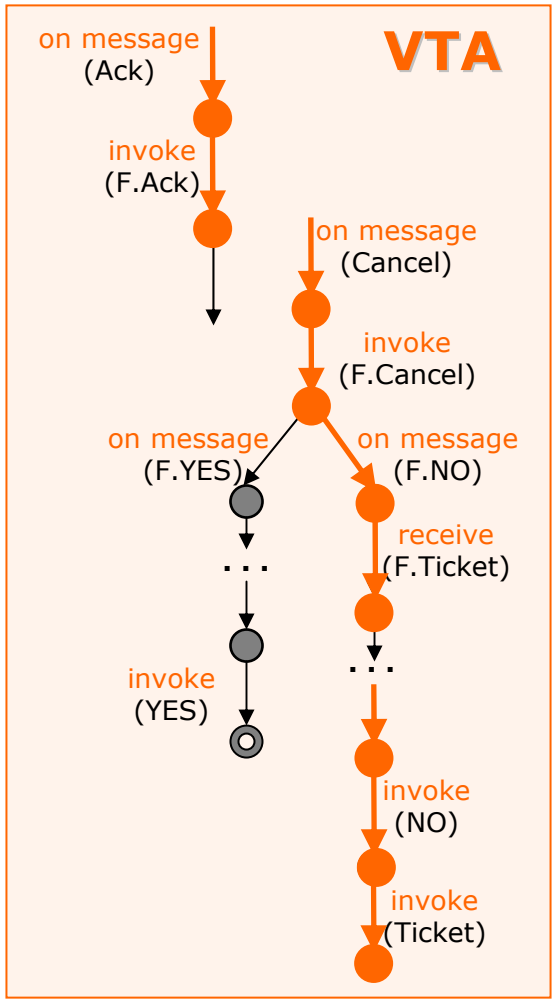


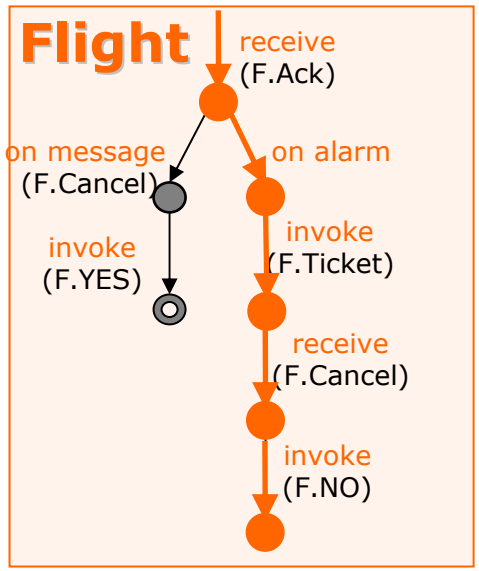
Deadlock?

No!

VTA Processes – Complex Cancellation







Composition Properties

- The ordered communication model is violated
  - The order of emissions of messages *F.Ticket* and *F.NO* is different from the order of receptions
- The real execution is be correct
  - Engines support message overpasses
  - In the scenario, all message are eventually consumed, and cancellation is performed correctly
- Also in this case, an **adequate communication model** is needed to model and formally verify this scenario!
- **Asynchronous unordered** communication model
  - No restrictions on the message order
  - Intuitively, one queue for each message type (very similar to real implementations)

Our approach

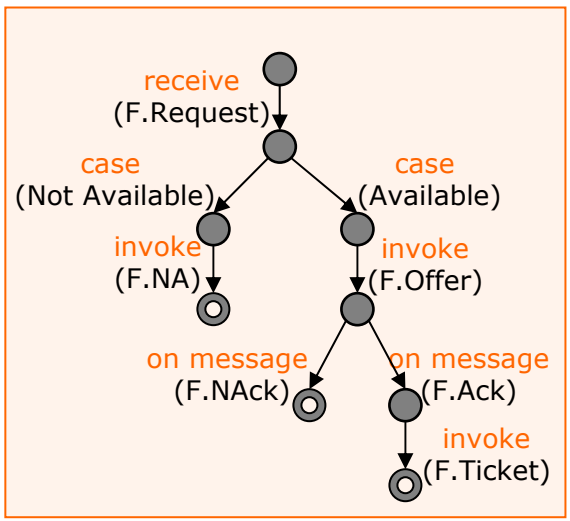
- Define a **set of communication models**
  - Different levels of complexity
  - Different interaction mechanisms
  - Common framework
- Given a certain composition scenario determine an **adequate** communication model
  - Represents all real executions of the composition
  - Preserves behavioral properties
- **Incremental** analysis process
  - From simpler to complex communication models
  - Check if the communication model is adequate w.r.t. the scenario
  - If yes, perform the formal verification against this model

Our approach: formal definitions

- Three main ingredients:
  - Component services are formally modeled as **State Transition Systems**
  - The modalities of the communications are formalized as a **Communication Model**
  - The composite behavior of the component services according to a specific communication model is formally described as a **Global State Transition System**

From BPEL to STS

- **State Transition System** $\Sigma = \langle S, S_0, I, O, R \rangle$ where
  - S – finite set of states
  - S_0 – set of initial states
  - I – set of input actions
  - O – set of output actions
  - $R \subseteq S \times (I \cup O \cup \{\tau\}) \times S$ – transition relation




```
PROCESS Flight
STATES {Start, switch_IsAvailable, OUT_FNA, SUCCESSS,…}
INPUT FRequest, FNAck, FAck
OUTPUT FNA, FOffer, FTicket
INIT
      state = Start
TRANS
      Start – [IN FRequest] -> switch_IsAvailable
      switch_IsAvailable – [TAU] -> OUT_FNA
      switch_IsAvailable – [TAU] -> OUT_FOffer
      …
```

Communication Model

- A communication model **Δ** is defined by a set of queues

  $\langle Q_1, Q_2, \ldots, Q_n \rangle$

  where each queue Q_i has associated:
  - A set of messages M_i
  - A (finite or infinite) bound B_i on the messages it can contain

- **Synchronous** communication model:
  - A single queue with bound 1

- **Ordered asynchronous** communication model:
  - One unbounded queue for (the messages exchanged between) each pair of services

- **Unordered asynchronous** communication model:
  - One unbounded queue for each message type

Communication Model

- A communication model **Δ** is defined by a set of queues

  $\langle Q_1, Q_2, \ldots, Q_n \rangle$

  where each queue Q_i has associated:
  - A set of messages M_i
  - A (finite or infinite) bound B_i on the messages it can contain

- Other communication models are possible:
  - One queue per process (**locally ordered model**): see paper
  - Mixed synchronous and asynchronous communications (e.g., manage only tickets/cancellations as asynchronous communications)
  - Mixed bounded/unbounded queues
  - …

Global State Transition System

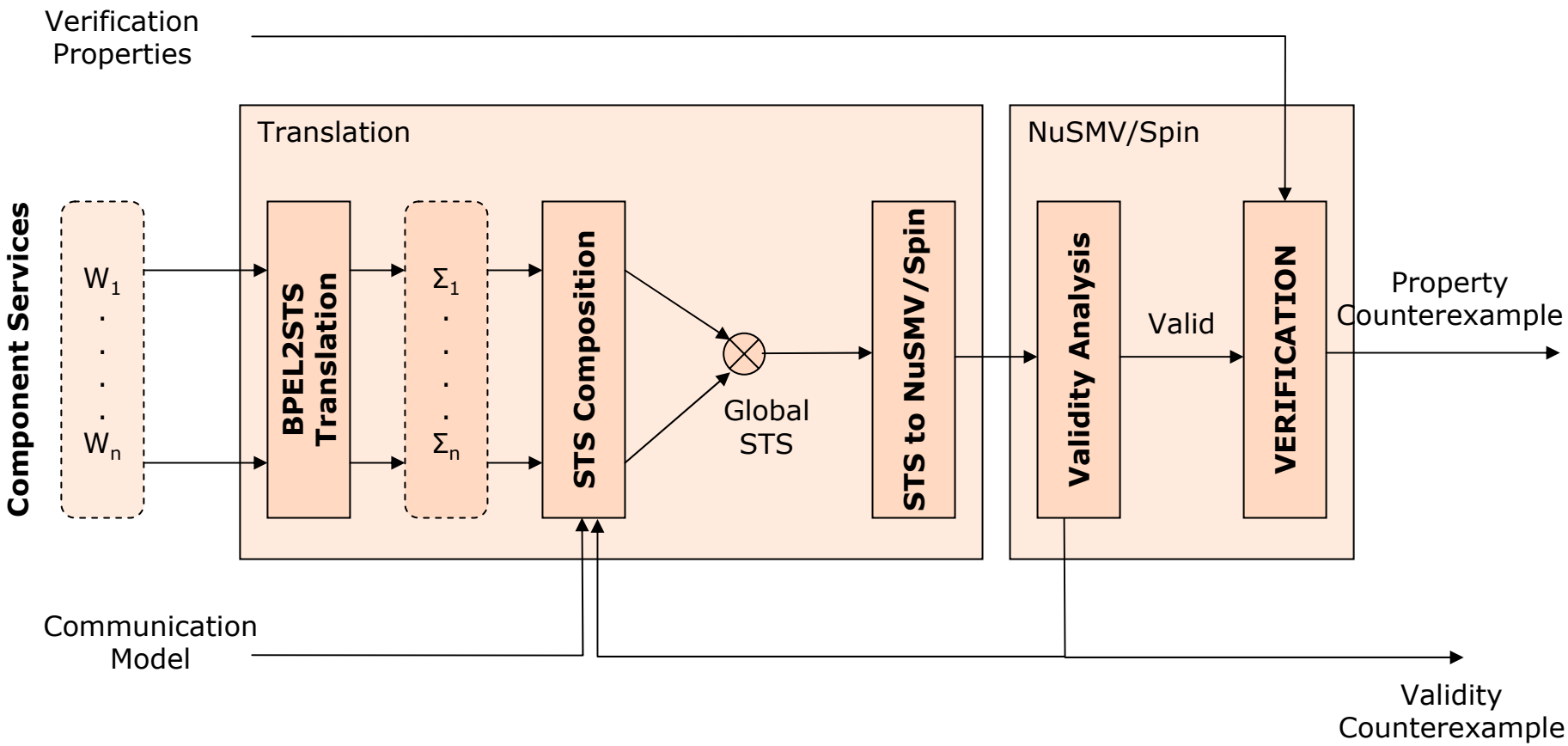
- A Global State Transition System (GSTS):
  - defines the composite behavior of the system.
  - is parametric wrt a communication model **Δ**

- A GSTS is a tuple $G = \langle GS, GS_0, A, T \rangle$, where:
  - GS are the global states; each state has the form $gs = (\langle s_1, s_2, \ldots, s_n \rangle, \langle q_1, q_2, \ldots, q_m \rangle)$, where:
    - s_i is the state of the i-th component STS
    - q_j describes the content of the j-th queue
  - GS_0 are the initial global states
  - A are the input-output actions
  - $T \subseteq GS \times A \times GS$ is the transition relation

GSTS: transitions

- The transition relation $T \subseteq GS \times A \times GS$ is defined as follows:
  - If the i-th STS performs an output:
    - update the status of the STS
    - add the emitted message to the associated queue
  - If the i-th STS performs an input:
    - consume a message from the associated queue (the queue has to be non-empty!)
    - update the status of the STS
  - If the i-th STS performs a TAU action:
    - update the status of the STS

Implementation



Experiments

- Instances of VTA Case Study
- Different communication models
- Different verification properties: deadlock, LTL properties
  - (F User.state = SUCCESS) ↔ ((F Hotel.state = SUCCESS) & F (Flight.state = SUCCESS))

Instance	Model	Translation	Validity	Deadlock	LTL
Example 1	Sync.	0.5 sec	1 sec (Valid)	0.5 sec	0.5 sec
Example 2	Sync.	2 sec	4 sec (Invalid)	–	–
	Ordered	4 sec	3 sec (Valid)	3 sec	2 sec
Example 3	Sync.	4 sec	5 sec (Invalid)	–	–
	Ordered	8 sec	7 sec (Invalid)	–	–
	Unordered	11 sec	6 sec (Valid)	5 sec	4 sec

Conclusions

- An **unified framework** for the analysis of Web service compositions under different communication models is presented
- The framework allows for **validation** of the composition against communication model and for **verification** of the valid composition
- A prototype **tool** based on the framework is implemented
- Future works:
  - Better support for **data** (application of "knowledge level" and "abstraction based" reasoning techniques)
  - **Conformance** problem: verify a **WS-BPEL** process against a **WS-CDL** choreography specification

Any question?