



# Towards the Verification of Services Collaboration

---

Jing Liu

East China Normal University



# Outline

---

- Motivation and Introduction
- Checking Consistency of Services
- Case Study
- Conclusion and Future Work



# Motivation and Introduction(1)

---

- Recently Service Oriented Architecture (SOA) attracts more and more attention as a promising technique
- SOA's feature of *loosely coupled* promises rapid services composition and dynamic reconfiguration



## Motivation and Introduction(2)

---

- However it also brings more challenges to the assurance of *consistency and reliability*
  - The whole view of service flow is invisible because most SOA applications compose various services from different runtime environments
  - There are many complicated interactions between services of a SOA application



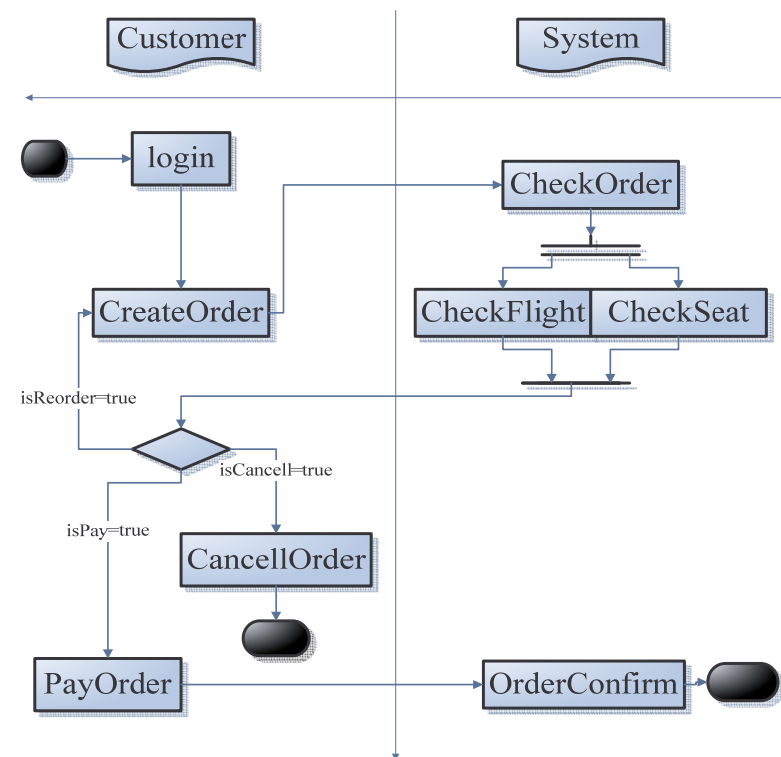
# Motivation and Introduction(3)

---

- Many methods and tools have been developed aiming to handle this problem
  - Component reuse
  - Behavior formalization
- In this paper, we propose an approach to verifying the consistency between collaborative services in SOA applications based upon model checking
  - Define Collaboration-Contracts (Discuss later) of collaborative services from an Extended UML Sequence Diagram
  - Use LTL Formula to present Collaboration-Contracts
  - Translate dynamic behavior models (From StateChart) into formal modeling specification (Promela for SPIN)
  - Verify the consistency of collaborative services through an integrated SPIN-binding tool called TMDA

# Motivation and Introduction(4)

- This workflow of a flight-ordering system implies some kind of relationship between services like pattern of “**send and receive**”
  - For example: We use *OCheck* (System checks flights and seats) and *PRequest* (Customers cancel the order) to represent the related transitions  
 $OCheck \wedge (isPay == true) \rightarrow PRequest$



# Checking Consistency of Services

## — Extending Sequence Diagram(1)

- ***Why UML?***

- Both UML Sequence and StateChart Diagrams represent model's dynamic behaviors
- StateChart Diagrams can unambiguously represent internal behavior of services
- Sequence Diagrams naturally imply sequential transitions between collaborative services

- ***Why extending UML?***

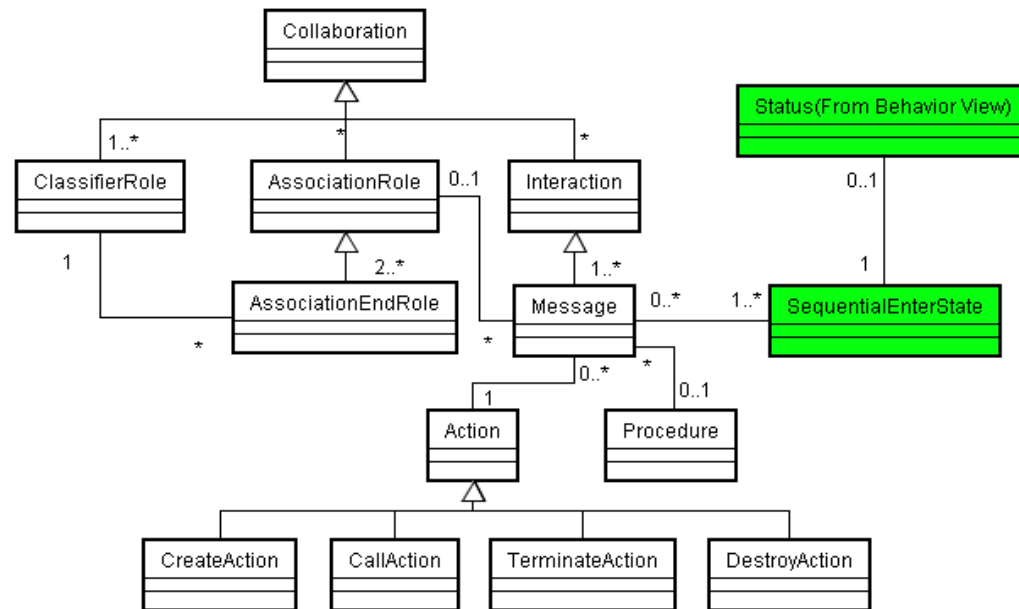
- Highly enhance modeling veracity and efficiency, as well as reduce possible inconsistency
- UML 2.0 does not explicitly unify naming rules to involve two diagrams

# Checking Consistency of Services

## — Extending Sequence Diagram(2)

- *Sequential Enter State*

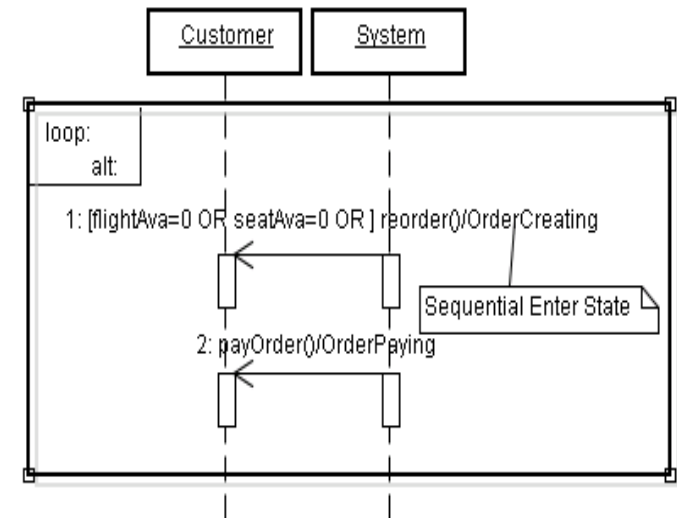
- An extended state property of messages (in UML Sequence Diagram), representing that the accepting service should enter the relevant state (in UML StateChart Diagram) after a message has been executed



# Checking Consistency of Services

## — Extending Sequence Diagram(3)

- **Sequence Enter State (SES)** can also be understood as snapshots of system's behavior status
- ***OrderCreating*** is the SES of message ***reorder***, which means when the service ***Customer*** has executed the message ***reorder*** from ***System***, ***Customer*** will enter the state ***OrderCreating*** at the very moment



# Checking Consistency of Services

## — Semantics of ESD(1)

**Definition [ESD Message]** A message in ESD is a quintuple

$EM = \langle SO, TO, M(), SES, Sn \rangle$  where

- $SO$  : the *source object* of the message;
- $TO$  : the *target object* of the message;
- $M()$  :  $g \rightarrow m$ , where  $g$  is the guard condition of  $m$ , and  $m$  can either be a method call like  $TO.m()$  or an action like create, terminate or destroy;
- $SES$  :  $TO$ 's  $SES$  after the message has been executed;
- $Sn$  : the serial number used to record the sequence of messages.

# Checking Consistency of Services

## — Semantics of ESD(2)

**Definition [Participant]** A participant is a triple  $P = \langle O, G, CS \rangle$  where

- $O$  : the object corresponding to the relevant participant;
- $G$  : the set of Boolean attribute *expressions*( $g$ ) of source object  $O$ , represents preconditions of  $O$ 's messages;
- $CS$  : the set of current states of  $O$  through the entire process, which is used to present dynamic behavior;

**Definition [ESD]** An ESD is a triple  $ESD = \langle Ps, EMs, IO \rangle$  where

- $Ps$  : the set of *Participants*;
- $EMs$  : the set of *ESD messages*;
- $IO$  : interaction operators, consist of global variables and expressions like *alt*, *opt*, *loop* etc, which is used to present relationship between messages;

# Checking Consistency of Services

## — Semantics of ESD(3)

- So the dynamic semantics of an ESD message can be defined as follows:

$EM(SO, TO, M(), SES, S_n)$  def= if  $SO \in P_s \wedge TO \in P_t$   
then  $P_s.g \wedge TO.m() \rightarrow P_t, CS = SES$  else false,  
where  $P_s, P_t$  represent relevant participates of  $SO$  and  
 $TO$ , respectively.

# Checking Consistency of Services

## —— Collaboration-Contracts(1)

- Collaboration-Contracts are restrictions of SESs from an ESD

**Definition [Collaboration-Contracts]** a four-tuple  $CC = \langle I, E, SESs, IO \rangle$  where

- $I$ : the first element of relevant  $SS$ ;
- $E$ : the last element of relevant  $SS$ ;
- $SESs$ : the set of  $SESs$  abstracted from  $SS$  in ESD;
- $IO$ : interaction operators, consists of global variables and expressions like alt, opt, loop etc. ( $SS$  represents set of sequential behavior states among collaborative services)

# Checking Consistency of Services

## —— Collaboration-Contracts(2)

- Dynamic Semantics of Collaboration-Contracts (CC)

$$CC[g, SES_1, SES_2] = g \wedge SES_1 \rightarrow g' \wedge SES_2$$

$$CC[opt(g, SES)] = g \rightarrow SES \wedge g'$$

$$CC[alt(g, SES_1, SES_2)] = (g \rightarrow SES_1 \wedge g_1) \vee (\neg g \rightarrow SES_2 \wedge g_2)$$

$$\vee (\neg g \rightarrow SES_2 \wedge g_2)$$

$$CC[loop(g, SES)] = (\exists n, g \rightarrow SES \wedge g')$$

$$\vee (\neg g \rightarrow SES \wedge g'), \text{ where } g' \text{ represents post-condition relevant to } g.$$

# Checking Consistency of Services

## —— Generating LTL Formula

- With the dynamic semantics defined above, we can use LTL formula to represent relevant CC easily.

$$I \wedge F E_{\neq}$$

We denote  $CC_i, CC_j$  as two continuous snapshots of a CC whose SESs are relevant to CSs of participates  $P_i, P_j$ . The LTL formula can be defined as follows:  $\neq$

$$P_i.g \wedge CC_i \rightarrow NCC_j, \text{ where } i < j, \neq$$

# Checking Consistency of Services

## — Generating Behavior Model

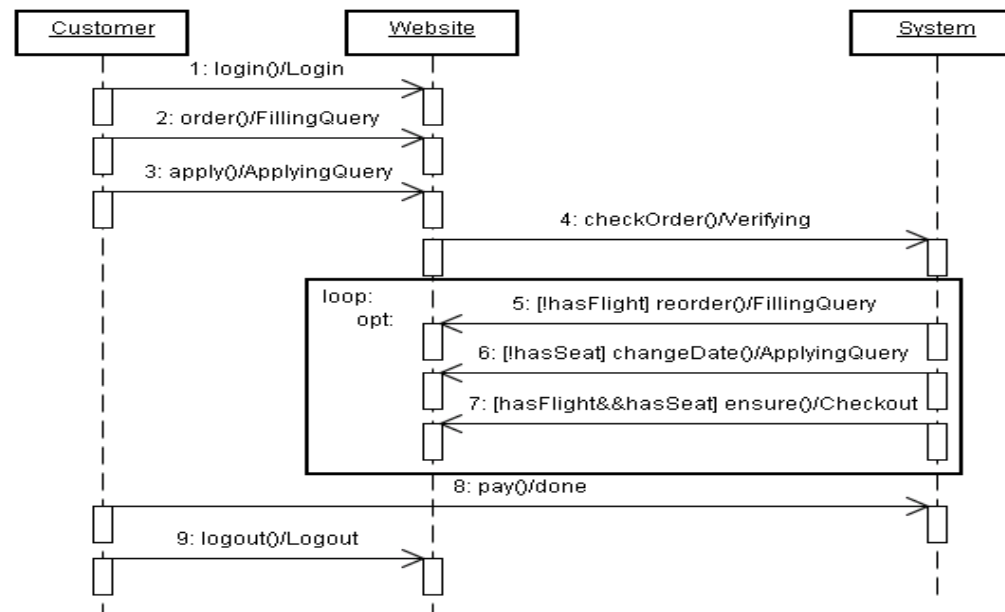
- **Mapping from StateChart Diagram to Promela**
- **Fork/Join can be represented by simple state transitions**
- **Composite State can be decomposed into several hierarchical states**

Meta Model of StateChart <sup>Ⓟ</sup>	Meta Model of Promela <sup>Ⓟ</sup>
State Diagrams: <sup>Ⓟ</sup> D1,D2 <sup>Ⓟ</sup>	Proctype Expression: <sup>Ⓟ</sup> active Proctype D1(),active Proctype D2() <sup>Ⓟ</sup>
State Node: S1,S2,InitialState,FinalState <sup>Ⓟ</sup>	Bit Expr: <sup>Ⓟ</sup> bit S1,S2, InitialState,FinalState <sup>Ⓟ</sup>
Variable Expression: <sup>Ⓟ</sup> bool/ int/short/unsigned/ enum <sup>Ⓟ</sup>	Promela DataTypes: bool/ int/short/unsigned/ mtype <sup>Ⓟ</sup>
Initial State Node <sup>Ⓟ</sup>	Run Proctypes <sup>Ⓟ</sup>
State Transition Edge: T1(S1,S2) && S1 is active, <sup>Ⓟ</sup> S1,S2 represent T1's Exit State and Enter State <sup>Ⓟ</sup>	Assignment: <sup>Ⓟ</sup> S1=1->S1=0,S2=1 <sup>Ⓟ</sup>
Final State Node: <sup>Ⓟ</sup> FinalState <sup>Ⓟ</sup>	Printf info and Break: printf("exit"),FinalState ->break <sup>Ⓟ</sup>
Interaction(explained by Fig 4) <sup>Ⓟ</sup>	Channel expression <sup>Ⓟ</sup>
Fork Node <sup>Ⓟ</sup>	Remove before mapping(discuss later) <sup>Ⓟ</sup>
Join Node <sup>Ⓟ</sup>	Remove before mapping(discuss later) <sup>Ⓟ</sup>
Composite State Node <sup>Ⓟ</sup>	Decompose <sup>Ⓟ</sup>

# Case Study

## — ESD of FTOS

- FTOS is a web-based SOA application consists of two services: *website handling* and *system background processing*. We call them *Web* and *Sys* for short here. ESD of FTOS is shown as follows:



# Case Study

## — Collaboration-Contracts of FTOS

- FTOS's *Collaboration-Contracts* can be defined as follows:

$I = \{Login\}$

$E = \{Logout\}$

$SESSs = \{Login, FillingQuery, ApplyingQuery, Verifying, IO, done, Logout\}$

$IO = \{loop(opt(\neg hasFlight, FillingQuery), opt(\neg hasSeat, ApplyingQuery), opt(hasFlight \wedge hasSeat, Checkout))\}$

# Case Study

## —— LTL Formula of FTOS

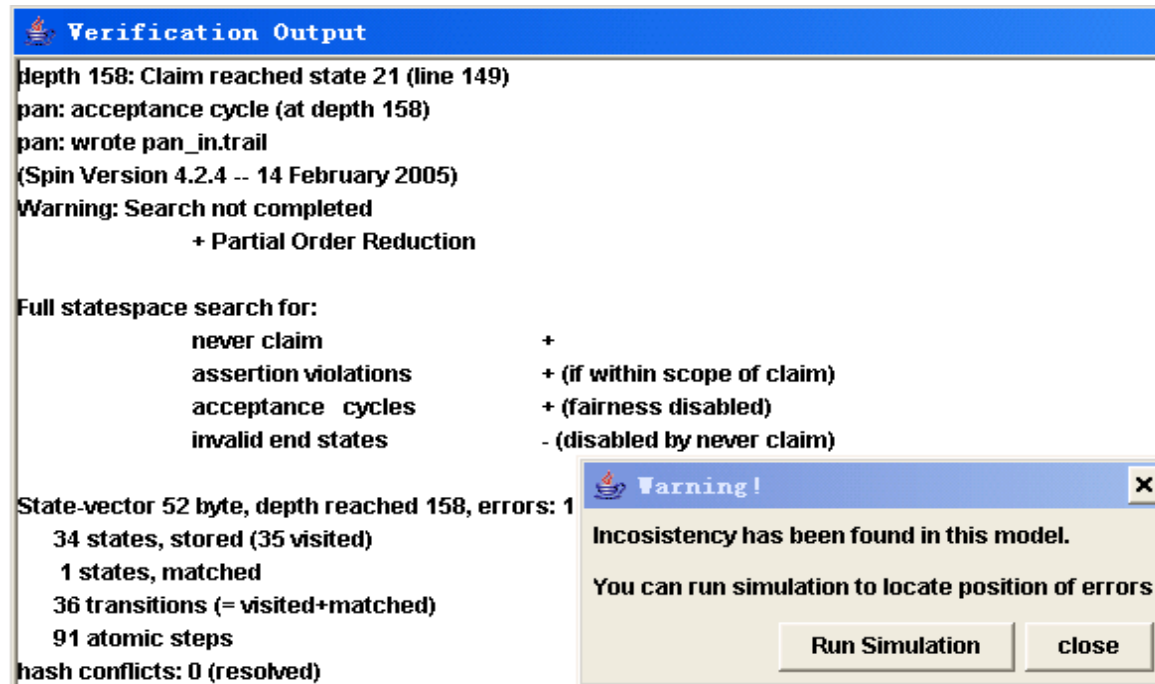
- Relevant *LTL* Formula can be defined:

$$\begin{aligned} & \text{Login} \wedge \mathbf{F} \text{Logout}; \cup \\ & (\text{Login} \rightarrow \mathbf{N} \text{FillingQuery}) \cup \\ & \wedge (\text{FillingQuery} \rightarrow \mathbf{N} \text{ApplyingQuery}) \cup \\ & \wedge (\text{ApplyingQuery} \rightarrow \mathbf{N} \text{Verifying}) \cup \\ & \wedge ((\text{Verifying} \wedge \neg \text{hasFlight} \rightarrow \mathbf{N} \text{FillingQuery}) \cup \\ & \quad \vee (\text{Verifying} \wedge \neg \text{hasSeat} \rightarrow \mathbf{N} \text{ApplyingQuery}) \\ & \quad \vee (\text{Verifying} \wedge \text{hasFlight} \wedge \text{hasSeat} \rightarrow \mathbf{N} \\ & \quad \text{Checkout})) \cup \\ & \wedge (\text{IO} \rightarrow \mathbf{N} \text{done}) \cup \\ & \wedge (\text{done} \rightarrow \mathbf{N} \text{Logout}); \cup \end{aligned}$$

# Case Study

## — Verification of FTOS

- *SPIN* is used to check the consistency among service behavior models, including *safety* and *liveness*. If the behavior model does not satisfy its *LTL* formula generated from *CC*, the violation traces will be shown by *SPIN* in the view of behavior model. Following is an output for *FTOS*.



The image shows a screenshot of a SPIN verification output window and a warning dialog box. The output window, titled "Verification Output", displays the following text:

```
Depth 158: Claim reached state 21 (line 149)
pan: acceptance cycle (at depth 158)
pan: wrote pan_in.trail
(Spin Version 4.2.4 -- 14 February 2005)
Warning: Search not completed
      + Partial Order Reduction

Full statespace search for:
      never claim          +
      assertion violations + (if within scope of claim)
      acceptance cycles   + (fairness disabled)
      invalid end states   - (disabled by never claim)

State-vector 52 byte, depth reached 158, errors: 1
  34 states, stored (35 visited)
   1 states, matched
  36 transitions (= visited+matched)
  91 atomic steps
hash conflicts: 0 (resolved)
```

Overlaid on the bottom right of the output window is a "Warning!" dialog box with the following text:

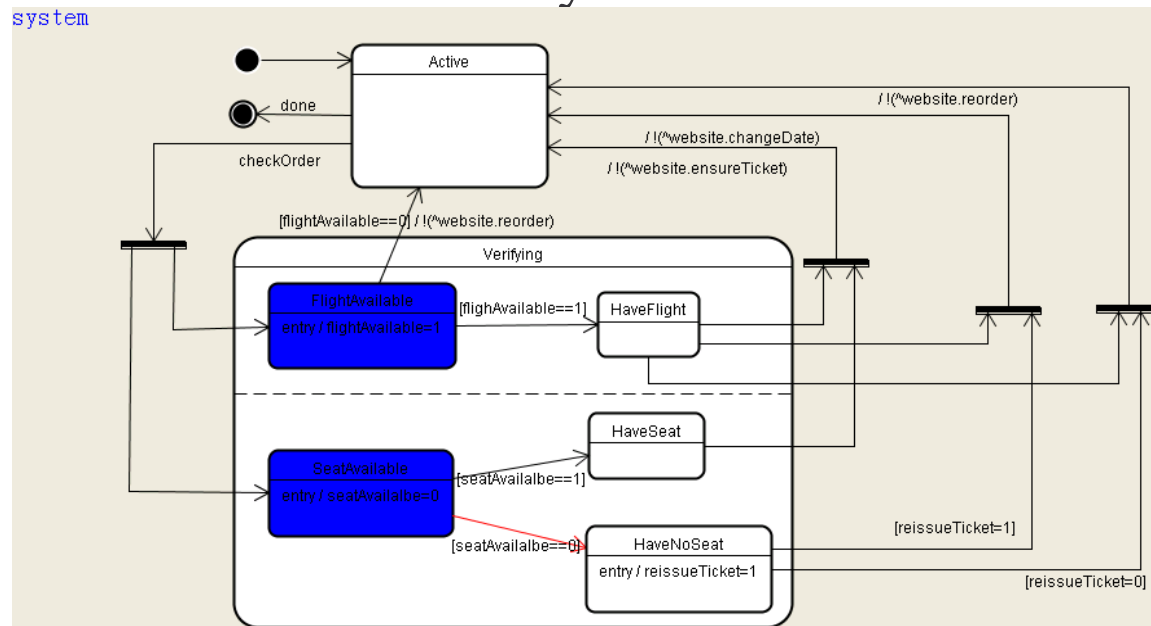
```
Warning!
Inconsistency has been found in this model.
You can run simulation to locate position of errors

[Run Simulation] [close]
```

# Case Study

## — Simulation of FTOS

- Customers can simulate transition of states step by step and observe system's snapshots of states changing, which is a great assist for inconsistency locating. A snapshot of *simulation* for *FTOS* by *TMDA* is shown as follow:





# Conclusion and Future Work(1)

---

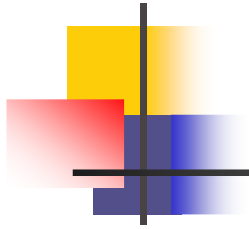
- This paper proposes an approach to verifying the consistency among collaborative services behavior based upon model checking .
- Compared to traditional SOA verification methods which are mostly designed at the level of code implementation, we introduce UML Diagrams to represent dynamic behavior among collaborative services aiming to reduce the risk and loss of SOA applications through the lifecycle of development
- Furthermore, this work provides an integrated SPIN-binding modeling tool (TMDA) for model-based verification and simulation. The tool has proven to be sufficiently flexible and expressive, and supports a variety of checks through the plug-in SPIN.



## Conclusion and Future Work(2)

---

- In the future, we will study systems built on top of services belong to different entities which requires accessing the internals of each service. Different data types between StateChart Diagrams and Promela also need be handled later



---

■ THANKS!