Modeling Web Services for Test Case Generation

Service Oriented Architectures Security

Ernesto Damiani

Università di Milano

Reference Scenario

- **Distributed and service-based infrastructure that includes three parties**
  - a ***customer*** accessing a service provided by a supplier
  - a ***supplier*** implementing and providing a service by exposing its interfaces through the Web
  - one or more ***third parties*** that have a business relationship with the supplier and provide a set of services that are used by the supplier to implement its business process

Reference Scenario: e-Bank

- **Supplier provides an e-Bank service**
- **Customers check their account, make fund transfer, and pay taxes**
- **Interfaces**
  - boolean subscription(*username*,*password*,*profile*)
  - boolean login(*username*,*password*)
  - result fundTransfer(*info*,*amount*)
  - result payTaxes(*info*,*amount*)
  - boolean confirm(*id*)
  - result getStatus(*account_id*)

Service implementation paradigms

***Single-call service*, a single interface is exposed and all activities are managed as supplier-internal computations**

***Conversation*, the supplier defines a WSCL file specifying the interactions with the customers in order to release the service**

***Orchestration or choreography*, services are composed by the supplier to implement its business process**

Web Service Modeling

- **Model of a service is the basis for test-based certification of service security properties**
  - Systems defined using a Symbolic Transition System (STS) model
  - Each transition regulated by a **guard**

- **Models are provided at different levels of granularity**
  - WSDL
  - WSCL
  - Test-based conditions (input-output constraints on variables)
  - Implementation details
  - WS-Policy

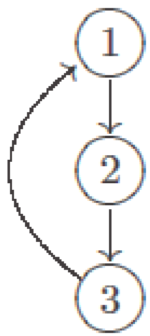
Web Service Modeling: Levels (1)

WSDL Interface only: A first basic STS-model considers the case in which a service provider exposes only the WSDL interface of its service

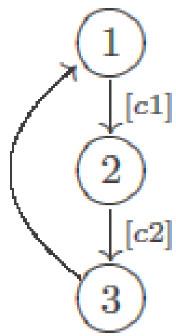
WSDL interface extended with test-based conditions: A service provider may be willing to expose WSDL interface enriched with test-based conditions on input and output calls

Extended WSDL interface and stateful implementation: The service provider may be willing to present the low-level stateful implementation of the service

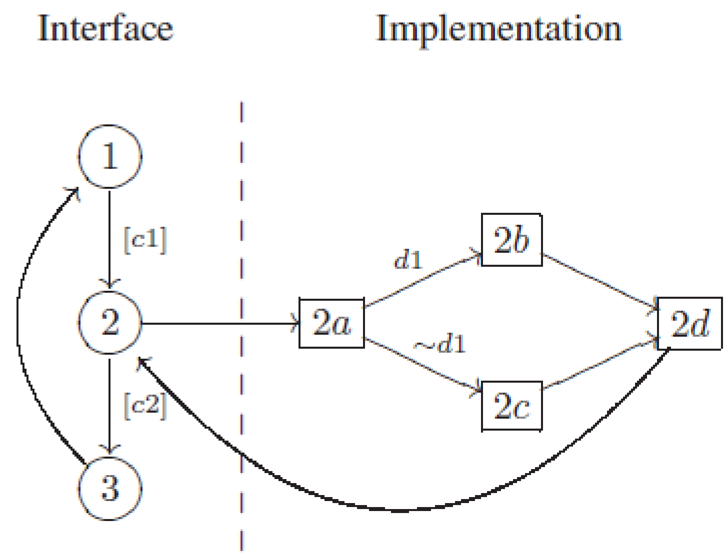
Web Service Modeling: Levels (2)



(a)



(b)



(c)

WSDL Interface only

WSDL with test-based conditions:

Extended WSDL and stateful implementation

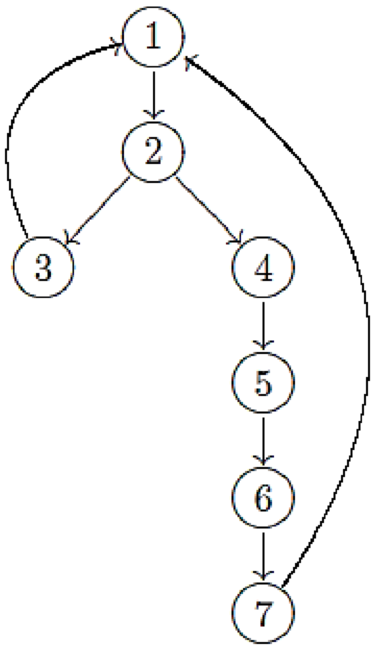
Web Service Modeling: Levels (3)

WSCL Interface only: The service provider may be willing to expose only the WSCL interface modeling the conversation with its customers

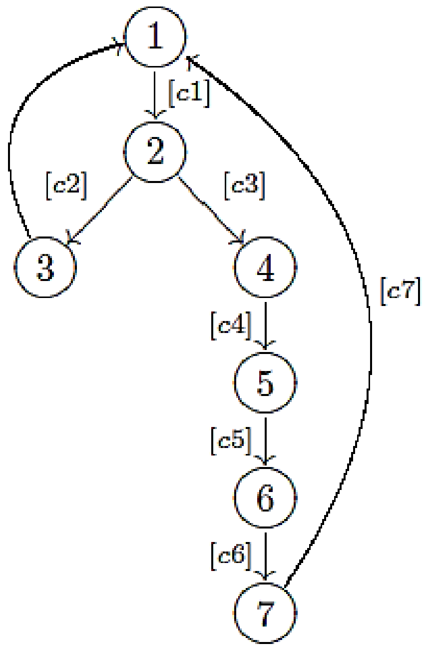
WSCL interface extended with test-based conditions: The service provider may be willing to expose WSCL interface enriched with test-based conditions on input and output calls

***Extended WSCL interface and stateful implementation*: The service provider may be willing to expose WSCL interface enriched with the overall stateful implementation**

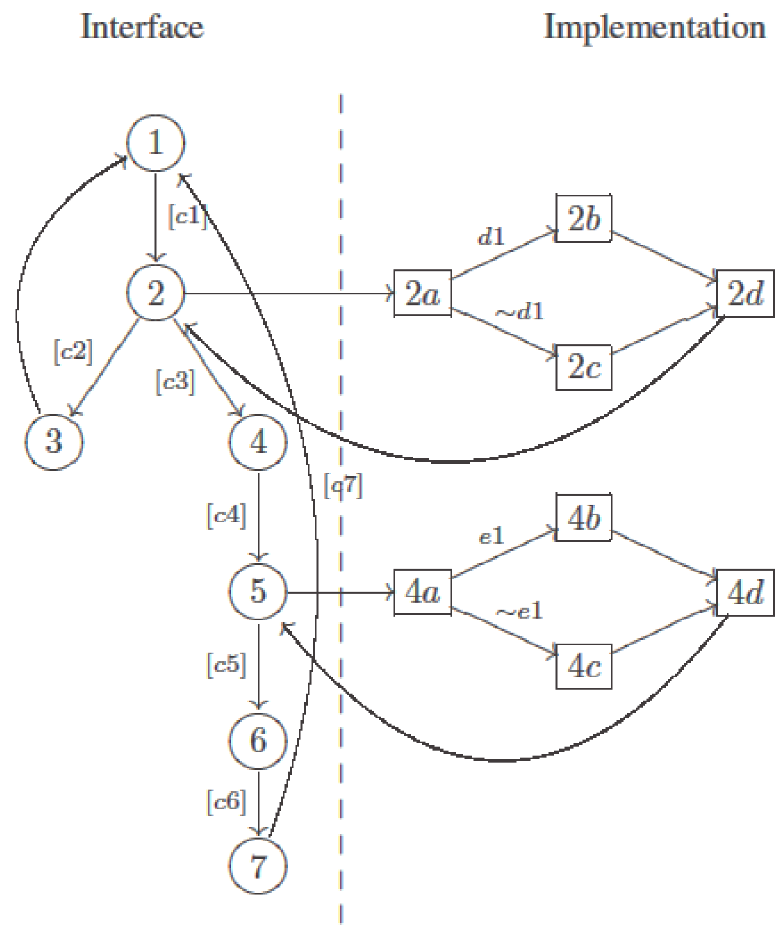
Web Service Modeling: Levels (4)



(a)



(b)



(c)

WSCL Interface only

WSCL with test-based conditions:

Extended WSCL and stateful implementation

Web Service Modeling: Levels (5)

Web service security specification: consideration of security services at container level

- Implemented on the top of the existing services
- Preserve integrity and confidentiality of the messaging
- Model of the service extended with hidden communication (e.g., key exchange)

An ordering is established between different models at different granularities

Models used to match and compare service certificates

Evidence-based Certification: A Model-based Approach

- **Model-based testing for service certification**

- **Need to specify in the certificate the amount of information (model) available at certification time**

- **The quality and effectiveness of the testing activities strongly depend on the available model**
  - E.g., WSDL-only permits black box testing, while implementation details permit white box testing with high coverage

bankTransfer *service: robustness against input malformation* (1)

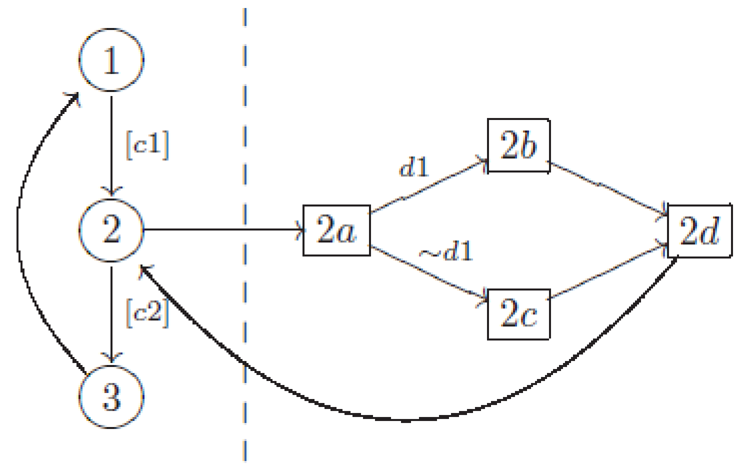
Simplified service that implements a single bankTransfer(*info*,*amount*) function

The model includes extended WSDL interfaces and stateful implementation

The customer calls bankTransfer(*info*,*amount*) and waits for the result

bankTransfer *service: robustness against input malformation* (2)

- **[c1] includes the call to the function bankTransfer and requires amount to be greater than zero and less than a max amount**
- **[c2] returns the output to the caller, and requires the amount in the result to be equal to the amount in the request, and the new balance to be equal to balance - amount or to be equal to 'error'**
- **[d1]:** ***balance≥amount***
- **[~d1]:** ***balance<amount***



bankTransfer *service: robustness against input malformation* (3)

PROPERTY: ROBUSTNESS
CLASS ATTRIBUTES: threat=*malformed input*

$$TC1 = \begin{cases} I: & (random)\ 0 \leq amount \leq max\_amount\ [c1] \\ & balance \geq\ amount\ [d1] \\ EO: & result.amount = amount \\ & result.balance = balance - amount \end{cases}$$

$$TC2 = \begin{cases} I: & (random)\ 0 \leq amount \leq max\_amount\ [c1] \\ & balance < amount\ [\sim d1] \\ EO: & result.balance = \text{`}Error\text{'} \end{cases}$$

$$TC3 = \begin{cases} I: & (random)\ amount \leq 0\ \vee\ amount \geq max\_amount[c1] \\ EO: & Fail \end{cases}$$

e-Bank *service: Integrity* (1)

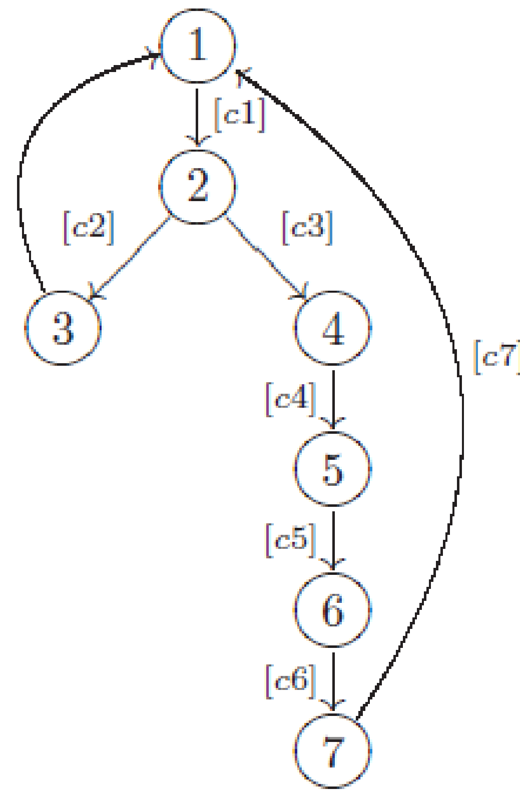
- **e-Bank service with a model including WSCL interfaces extended with test-based conditions**

- **The customer**
  1. logs in
  2. calls bankTransfer(*info*, *amount*) to make the transfer
  3. calls confirm(*id*) for the final confirmation

- **This model does not take into account the signature verification algorithm "provision is in place"**

- **A stronger certificate can be made on a model including internal signature checking**

- **Integrity can be certified at container level (WS-Security, WS-Policy)**



e-Bank *service: Integrity* (2)

PROPERTY: INTEGRITY
CLASS ATTRIBUTES: `algorithm`=*RSA*; digest=SHA-256; |key|=1024bit

$$TC1 = \begin{cases} I: & Message_i \ + \ Valid\ Signature \\ EO: & decrypt_{P_i}[signature] = digest[Message_i] \end{cases}$$

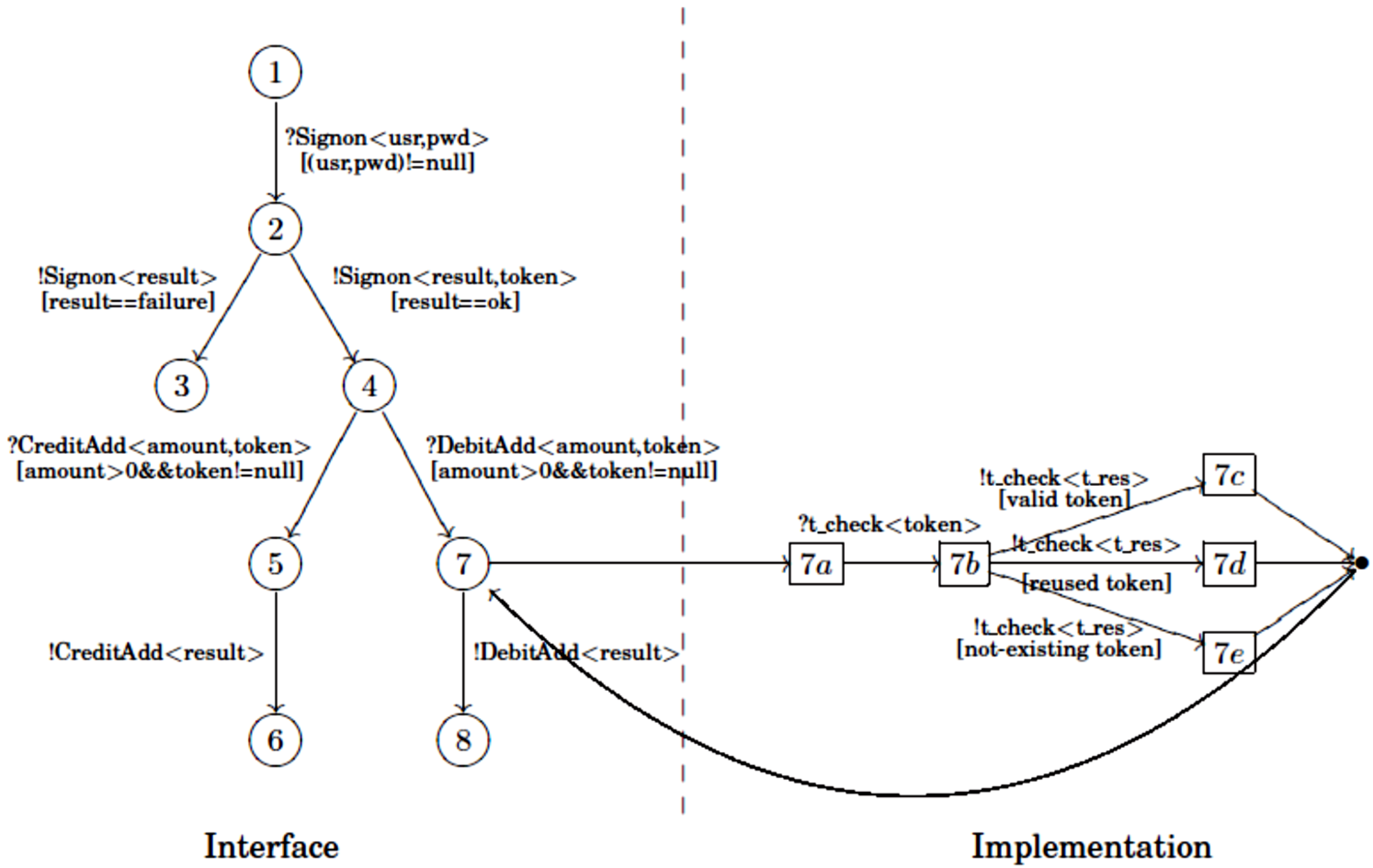
$$TC2 = \begin{cases} I: & Message_i \ + \ Invalid\ Signature \\ EO: & decrypt_{P_i}[signature] \neq digest[Message_i](fail) \end{cases}$$

Another Example: IFX-Based Reverse ATM Service

IFX-based *Deposit and Withdrawal* service

- Provides the typical functionalities of a reverse ATM for deposit and withdrawal
- It allows clients to remotely deposit/withdraw money in/from their bank account
- Clients can use a credit/debit card with PIN, or a username and password to authenticate to the IFX-based service at the bank via the reverse ATM.

Another Example: IFX-Based Reverse ATM Service



Another Example: Test Cases

PROPERTY: AUTHENTICITY
CLASS ATTRIBUTES: *SF*=token+PWD, *environment*=trusted

$$TC4 = \begin{cases} I_1: & (usr, pwd) \in \boldsymbol{ACL} \\ EO_1: & result = \boldsymbol{ok} \; + \; \boldsymbol{token} \\ I_2: & 0 < amount \leq account\ balance \wedge \; token \\ EO_2: & result = \boldsymbol{ok} \end{cases}$$

$$TC5 = \begin{cases} I_1: & (usr, pwd) \in \boldsymbol{ACL} \\ EO_1: & result = \boldsymbol{ok} \; + \; \boldsymbol{token} \\ I_2: & amount > account\ balance \; \wedge \; token \\ EO_2: & result = \boldsymbol{failure} \end{cases}$$

$$TC6 = \begin{cases} I_1: & (usr, pwd) \in \boldsymbol{ACL} \\ EO_1: & result = \boldsymbol{ok} \; + \; \boldsymbol{token} \\ I_2: & 0 < amount \leq account\ balance \; \wedge \; (not\ existing\ token \vee \; reused\ token) \\ EO_2: & result = \boldsymbol{failure} \end{cases}$$

$$TC7 = \begin{cases} I_1: & (usr, pwd) \notin \boldsymbol{ACL} \; \wedge \; (usr, pwd) \neq \boldsymbol{null} \\ EO_1: & result = \boldsymbol{failure} \\ I_2: & 0 < amount \leq account\ balance \; \wedge \; valid\ token \\ EO_2: & result = \boldsymbol{failure} \end{cases}$$

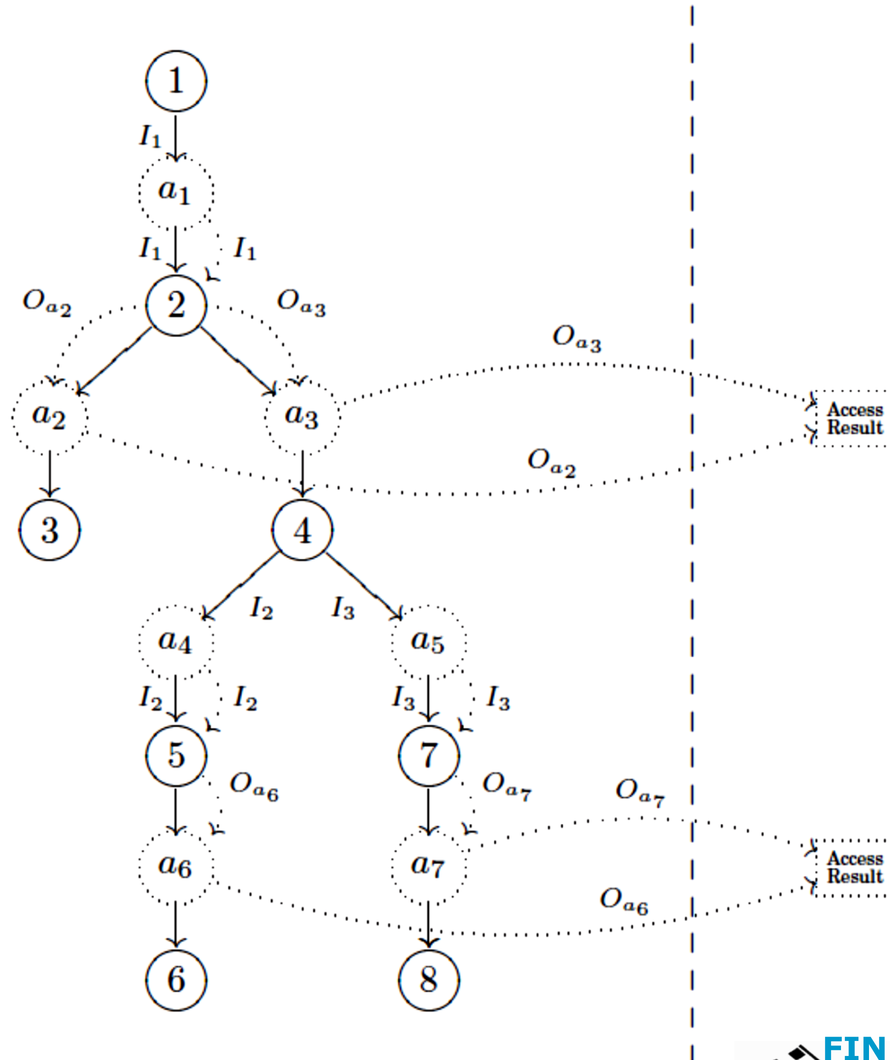
$$TC8 = \begin{cases} I_1: & (usr, pwd) \in \boldsymbol{ACL} \\ EO_1: & result = \boldsymbol{ok} \; + \; \boldsymbol{token} \\ I_2: & amount \leq 0 \; \vee \; token = null \\ EO_2: & result = \boldsymbol{failure} \end{cases}$$

$$TC9 = \begin{cases} I_1: & ((usr, pwd) \notin \boldsymbol{ACL} \; \wedge \; (usr, pwd) \neq \boldsymbol{null}) \vee \; usr = null \vee \; pwd = null \\ EO_1: & result = \boldsymbol{failure} \\ I_2: & amount \leq 0 \; \vee \; token = null \\ EO_2: & result = \boldsymbol{failure} \end{cases}$$

An Example of STS-Based Model for Penetration Testing

STS-based model for a replay attack

- Black nodes and lines model real communications
- Dotted lines are under the control of the attacker and represent the real attack implementation



FINE