

Verification of Service Oriented Architectures with ATP and SMT

S. Ranise

Università di Verona

Theory Engineering: Tools and Practice
(Cambridge - Feb. 8 and 9, 2010)



AVANTSSAR (Automated VALIDation of Trust and Security of Service-oriented ARchitectures)

- Strep EU project: www.avantssar.eu
- Consortium:
 - ▶ University of Genova
 - ▶ INRIA-Lorraine, Nancy
 - ▶ ETH Zurich
 - ▶ SIEMENS AG, Munich,
 - ▶ SAP AG, Sophia-Antipolis
 - ▶ Institute e-Austria, Timisoara
 - ▶ OPENTRUST, Paris
 - ▶ IBM Research Lab, Zurich
 - ▶ IRIT, Toulouse
 - ▶ University of Verona (coordinator)
- Entered its 3rd and last year

1 Background

2 Case Study

3 Symbolic Execution

4 Conclusions

5 Bibliography

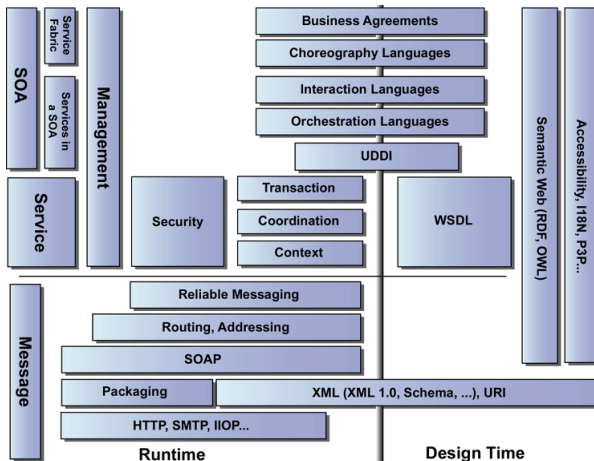
Services and Service Oriented Architectures (SOA)

- **Service** = artifact consisting of
 - ▶ identifier
 - ▶ interface
 - ▶ workflow
- Example: **web service** = software service accessible via standardized protocol
 - ▶ identifier: **URI**
 - ▶ interface: **WSDL**
 - ▶ workflow: **BPEL**
- **Service Oriented Architecture** = basis for software system design, inter-operation, and integration so as to move towards the
 - ▶ **Software as Service** paradigm

SOA Languages

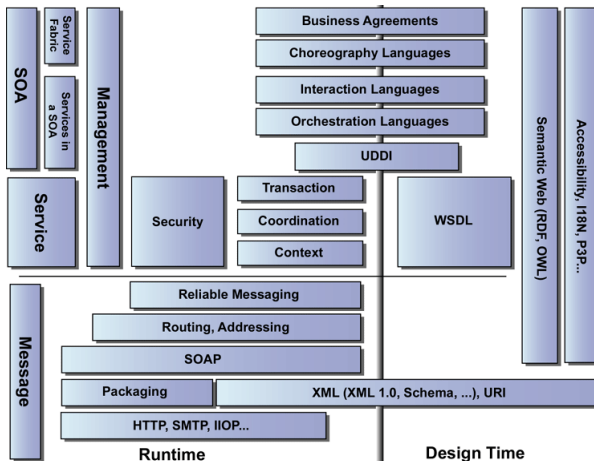
- **Variety of languages at different level of abstractions** for specification of SOAs
 - ▶ **Black-Box**: specification limited to signature information describing the sequences of messages supported by the interface [Web-Service Description Language (**WSDL**)]
 - ▶ **White-Box**: specification of the internal logic by high-level **workflow** formalism [Business Process Execution Language (**BPEL**)]
- **SOAs are data-intensive**
 - ▶ In most cases, **infinite** state models
- Large **semantic gap**: many popular languages lack formal semantics (e.g., BPEL)

An overview of SOA languages



Even worse...

An overview of SOA languages



Even worse...

Usually, there is more than workflow: Access Policies

- Operations of a service **can be executed** if this owns a **valid certificate** to do so
- Validity of certificates may **change over time**
- Example: virtual PC meeting

*Policy governing access to the reviews of a paper:
A reviewer assigned to a paper is required to submit
his/her own review before being able to read those
of the others*

To decide to **grant/deny** an **access request** to the reviews of a paper, the SO application needs to maintain and consult

- ▶ which reviewers have already submitted a review [**workflow**]
- ▶ identities and roles of various PC members [**policy management**]
- **Interplay** between the **workflow** and the **policy management!** ANTSSAR

A widespread design approach

- Two-level design space:
 - ① **separation** between **workflow** (WF) and **policy management** (PM)
 - ② **identification** of the interplay points between WF and PM
- Advantages:
 - ▶ enhance maintainability
 - ▶ reuse of policies
 - ▶ more precise/structured specifications
- **QUESTION**: can we mirror this way of structuring specifications of SO applications in a formal framework?

Our approach to specifying SOA

Extension of Manna and Pnueli's approach for distributed systems

- Static aspect
 - ▶ **First-Order Logic** (FOL) **formulae** to describe the (potentially infinite) **state space** of SO applications
 - ▶ **Catalogue** of FOL **theories** to describe widely used **data structures** (e.g., enumerated datatypes, lists, arrays)
 - ▶ **Standard ways** to **declaratively** represent **policies** as classes of FOL formulae (e.g., **Datalog**)
- Dynamic aspect
 - ▶ **FOL formulae** to describe **actions**/transitions of SO applications
 - ▶ **Uniform way** of expressing both the **data-flow** and the **workflow**
- Property
 - ▶ **Linear-Time** temporal **Logic** (LTL) formulae to describe correct/desired behaviors
 - ▶ Techniques for **reducing** the **verification** problem to **satisfiability** ANTSSAR problems in **FOL**

SOA verification

Given

- service specification $\mathcal{S} = (\text{identifiers, interface, workflow, policies})$
 - ▶ Workflow = (control, data)
 - ▶ Interplay even between control and data
- property P

We want to investigate the following verification problem:

do all/some executions of \mathcal{S} satisfy P ?

Our approach to verifying SOA

- Reduce verification problems to logical problems in **FOL**
- Use state-of-the-art ATP and/or SMT solving
- **Need** of **flexible**, **modular**, and **tunable** translation mechanisms
 - ▶ flexibility: to arrange the wide variety of SOA languages
 - ▶ modularity: to combine workflow and policy level specifications
 - ▶ tuning: to support/enhance mechanization
- To obtain this, the key ingredient is

FOL theory

- 1 Background
- 2 Case Study**
- 3 Symbolic Execution
- 4 Conclusions
- 5 Bibliography

Purchase Order

- Six activities involved:
 - ① creation of purchase order requesting goods from supplier
 - ② approval of purchase order prior to dispatch to supplier
 - ③ ack of delivery of the goods by signing goods-received note
 - ④ ack of delivery by countersigning goods-received note
 - ⑤ creation of payment file
 - ⑥ approval of the payment
- Partial order on activities: 1 followed by 2, 2 followed by either 3 or 5, 3 followed by 4, 5 followed by 6, and 4 followed by 6
- Policy level: modified RBAC model
 - ▶ users identified by digital credentials
 - ▶ user assigned to a role if the user's credentials match the user's attribute conditions associated with a role
 - ▶ authorization constraints, e.g. SoD

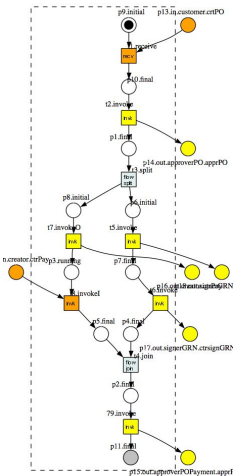
Purchase Order in BPEL

```
<process name="purchaseOrderProcess" ... >
```

```
...
```

```
<sequence>
  <receive operation="crtPO" variable="PO" </receive>
  <invoke operation="apprPO" inputVariable="PO" </invoke>
  <flow>
    <sequence>
      <invoke operation="signGRN" inputVariable="GRN" </invoke>
      <invoke operation="ctrsignGRN" inputVariable="GRN" </invoke>
    </sequence>
    <invoke operation="ctrPay" outputVariable="PF" </invoke>
  </flow>
  <invoke operation="apprPay" inputVariable="PF" </invoke>
</sequence>
</process>
```

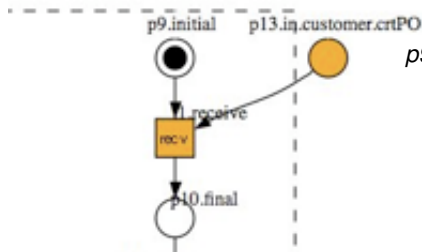
Purchase Order: BPEL process as a Petri net



Petri net generated from purchaseOrderProcess.bpel

Purchase Order: from Petri net to FOL with Linear Arithmetic

- If tokens are unbounded, use Integer variables



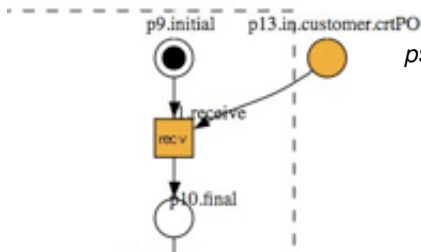
$$\begin{aligned}
 & p9.initial \geq 1 \wedge p13.in.customer.crtPO \geq 1 \wedge \\
 & p10.final' = p10.final + 1 \wedge \\
 & p9.initial' = p9.initial - 1 \wedge \\
 & p13.in.customer.crtPO' = \\
 & \quad p13.in.customer.crtPO - 1
 \end{aligned}$$

Automatically obtained by hacking the `bpe12owfn` tool

- Crude (counting-like) abstraction, we can do better but this requires... better hacking :-)

Purchase Order: from Petri net to FOL with Linear Arithmetic

- If tokens are unbounded, use Integer variables



$$\begin{aligned}
 & p9.initial \geq 1 \wedge p13.in.customer.crtPO \geq 1 \wedge \\
 & p10.final' = p10.final + 1 \wedge \\
 & p9.initial' = p9.initial - 1 \wedge \\
 & p13.in.customer.crtPO' = \\
 & \quad p13.in.customer.crtPO - 1
 \end{aligned}$$

Automatically obtained by hacking the `bpe12owfn` tool

- Crude (counting-like) abstraction, we can do better but this requires... better hacking :-)

Purchase Order: Policy Access in RBAC for BPEL

Permission	
p1	Approve purchase order
p2	Sign goods-received note
p3	Counter sign goods-rcv.
p4	Create payment file
p5	approve payment

Role	Permission
FinClerk	p4
FinAdmin	p5
POClerk	p3
POAdmin	p1

Authorization constraints

- the user that has approved the purchase order must be distinct from the one signing the goods-received note
- ...

Policy Access: from RBAC for BPEL to BSR fragment of FOL

- **Bernays-Schönfinkel-Ramsey** = $\exists\forall$ -prefix + quantifier-free formula with only predicate and constant symbols
- Satisfiability of BSR formulae is well-known to be decidable
- Easy to encode tables, e.g.

$$\forall P, A. perm(P, A) \Leftrightarrow \left(\begin{array}{l} (P = p1 \wedge A = \text{"Approve purchase order"}) \quad \vee \\ (P = p2 \wedge A = \text{"Sign goods-received note"}) \quad \vee \\ \dots \end{array} \right)$$

- It is indeed possible to automatically translate table-based RBAC specifications to formulae in the BSR fragment

Policy Access: what about access constraints?

- Add **time dependent predicates**: *executed* and *can_exec*
- Extend transitions, e.g.

$$\exists U. \left(\begin{array}{l} p9.initial \geq 1 \wedge p13.in.customer.crtPO \geq 1 \wedge \\ can_exec(U, "Approve purchase order") \wedge \\ p10.final' = p10.final + 1 \wedge \\ p9.initial' = p9.initial - 1 \wedge \\ p13.in.customer.crtPO' = p13.in.customer.crtPO - 1 \wedge \\ \forall X, Y. executed'(X, Y) \Leftrightarrow \left((X = U \wedge Y = "Approve purchase order") \vee \right. \\ \left. executed(X, Y) \right) \end{array} \right)$$

Policy Access: what about access constraints?

- Add **time dependent predicates**: *executed* and *can_exec*
- Extend transitions, e.g.

$$\exists U. \left(\begin{array}{l} p9.initial \geq 1 \wedge p13.in.customer.crtPO \geq 1 \wedge \\ \text{can_exec}(U, \text{"Approve purchase order"}) \wedge \\ p10.final' = p10.final + 1 \wedge \\ p9.initial' = p9.initial - 1 \wedge \\ p13.in.customer.crtPO' = p13.in.customer.crtPO - 1 \wedge \\ \forall X, Y. \text{executed}'(X, Y) \Leftrightarrow \left((X = U \wedge Y = \text{"Approve purchase order"}) \vee \right. \\ \left. \text{executed}(X, Y) \right) \end{array} \right)$$

Policy Access: what about access constraints? (cont'd)

- Define access constraints, e.g.

$$\forall X, Y. can_exec(U, A) \Leftrightarrow \left(\left(A = \text{"Sign goods-received note"} \wedge perm(U, A) \wedge \exists U'. (U' \neq U \wedge executed(U', \text{"Approve purchase order"})) \right) \vee \dots \right)$$

- can_exec* is time variant because it is defined in terms of *executed*
- This is still a BSR formula under the assumption that users are finitely many and known a priori (this is a reasonable assumption when considering scenarios) as the existential quantifier can be transformed to a (finite) disjunction

Policy Access: what about access constraints? (cont'd)

- Define access constraints, e.g.

$$\forall X, Y. can_exec(U, A) \Leftrightarrow \left(\left(A = \text{"Sign goods-received note"} \wedge perm(U, A) \wedge \exists U'. (U' \neq U \wedge executed(U', \text{"Approve purchase order"})) \right) \vee \dots \right)$$

- can_exec* is time variant because it is defined in terms of *executed*
- This is still a BSR formula under the assumption that users are finitely many and known a priori (this is a reasonable assumption when considering scenarios) as the existential quantifier can be transformed to a (finite) disjunction

Policy Access: what about access constraints? (cont'd)

- Define access constraints, e.g.

$$\forall X, Y. can_exec(U, A) \Leftrightarrow \left(\left(A = \text{"Sign goods-received note"} \wedge perm(U, A) \wedge \exists U'. (U' \neq U \wedge executed(U', \text{"Approve purchase order"})) \right) \vee \dots \right)$$

- can_exec is time variant because it is defined in terms of $executed$
- This is **still a BSR formula under the assumption that users are finitely many and known** a priori (this is a reasonable assumption when considering scenarios) as the existential quantifier can be transformed to a (finite) disjunction

- 1 Background
- 2 Case Study
- 3 Symbolic Execution**
- 4 Conclusions
- 5 Bibliography

Executability

- Given high non-determinism and subtle interplay between WF and PM levels, **executability of SO applications scenarios is unclear**
- Let τ_0, \dots, τ_n be a sequence of (partially instantiated) transition formulae and $\varphi(\underline{x}_0, \underline{p}_0)$ be a formula defining the set of initial states

The **(symbolic) execution problem** consists of checking

$$\varphi(\underline{x}_0, \underline{p}_0) \wedge \bigwedge_{i=0}^n \tau_i(\underline{x}_i, \underline{p}_i, \underline{x}_{i+1}, \underline{p}_{i+1}) \text{ is } T_{SOA}\text{-satisfiable}$$

where T_{SOA} is the combination of the theory for the workflow and the theory for the policy access

Executability

- In theory, we can study sufficient conditions for decidability
- In practice, we have implemented the approach in a tool called **WSSMT** (master project by Luca Zanetti)
 - ▶ Client-server architecture
 - ▶ Eclipse plug-in
 - ▶ Mixture of ATP and SMT technology
 - ▶ Input language: subset of DFG syntax (SPASS input syntax)
 - ▶ Interface to ATPs: `dfg2tptp` tool available in SPASS distribution
 - ▶ Interface to SMTs: `dfg2smtlib` tool which is an adaptation of the `dfg2tptp`

Using ATP in WSSMT

- Problem: how to handle arithmetic?
 - ▶ so far, only approximations ...
 - ▶ recently, some ATPs (e.g., Vampire, Otter, SPASS) started to natively support it and some standardization effort is going on
- Unpredictable behavior on satisfiable problem
- Surprisingly difficult to handle enumerated data-types

$$\bigwedge_{1 \leq i \neq j \leq n} c_i \neq c_j \quad (\text{at-least})$$

$$\forall x. (x = c_1 \vee \dots \vee x = c_n) \quad (\text{at-most})$$

- ▶ superposition often diverges on sets of clauses containing these axioms...
- ▶ SPASS developed a variant of superposition to handle this case ANTSSAR (not yet integrated in the main release)

Using SMT in WSSMT

- No problem in handling arithmetic
- More “predictable” behavior
 - ▶ Quickly terminate ...
 - ▶ Return `unknown` (when quantifiers are present)
 - ▶ Recent advance in handling quantifiers:
 - ★ Z3 has very good heuristics for quantifier instantiation coupled with decision procedure
 - ★ it even features an efficient BSR decision procedure, unfortunately not integrated with decision procedures
- Surprisingly, heuristics for quantifier handling have problems with previous axiomatization of enumerated data-type
 - ▶ Introduce new function casting elements to integers

$$\bigwedge_{i=1}^n f(c_i) = i$$

$$\forall x, y. (f(x) = f(y) \Rightarrow x = y)$$

- 1 Background
- 2 Case Study
- 3 Symbolic Execution
- 4 Conclusions**
- 5 Bibliography

Wish list

- flexibility: identify a set of useful theories which become targets of SOA languages (BPEL, RBAC, ...)
- modularity: combine workflow and access policy smoothly
- tuning: mechanical support for theory massaging, targeted to reasoners
- How to handle recursive policies?
 - ▶ Fix-point computation seems essential
 - ▶ Preliminary (but only theoretical) investigations: characterized a set of theories for which fix-point computation terminates
 - ▶ Integration of a Prolog engine?

On-going/future work

- **On-going**: formalization of several case-studies inspired by the industrial partners in the AVNTSSAR project (by Alberto Calvi, prospective PhD student)
- **On-going**: tuning of ATPs and SMT solvers for the class of proof obligations considered when formalizing BPEL processes with RBAC policies
- **Future**: smoother integration of `bpel2owfn` in WSSMT and extension to richer data flow (sending/receiving messages, updating variables, ...)

- 1 Background
- 2 Case Study
- 3 Symbolic Execution
- 4 Conclusions
- 5 Bibliography**

Some pointers to the literature (I)

- Formal framework with theoretical investigations (including invariant verification, not only symbolic execution)



M. Barletta, S. Ranise, and L. Viganò.

Verifying the Interplay of Authorization Policies and Workflow in Service-Oriented Architectures.

In Proc. of the 2009 International Symposium on Secure Computing (SecureCom 2009), Volume 3 of 2009 International Conference on Computational Science and Engineering (CSE 2009), pages 289–299. IEEE Computer Society Press, 2009.

<http://doi.ieeecomputersociety.org/10.1109/CSE.2009.172>

Some pointers to the literature (II)

- Preliminary investigation of the verification of access policies with recursion in the context of E-commerce



S. Ranise.

Towards Verification of Security-Aware Transaction
E-services, 2009.

In Proceedings of International Workshop on First-Order
Theorem Proving, Oslo, Norway, July 6-7 2009.

[http://www.mpi-inf.mpg.de/~sofronie/ftp09/
ftp09-proceedings.pdf](http://www.mpi-inf.mpg.de/~sofronie/ftp09/ftp09-proceedings.pdf)