Towards a Symbolic Semantics for Service-oriented Applications

Francesco Tiezzi

Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze

PLACES’08
International Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software

Oslo, Norway, - 7 June 2008

Joint work Rosario Pugliese and Nobuko Yoshida
Outline

1. Scenario
2. COWS
3. Verification problems
4. Symbolic semantics
5. Conclusion
Outline

1 Scenario
2 COWS
3 Verification problems
4 Symbolic semantics
5 Conclusion
Service-oriented computing & Web services

- **Service-oriented computing (SOC)**
  - an emerging paradigm for developing loosely coupled, interoperable, evolvable systems and applications

- **Web services** are a successful instantiation of SOC
Service-oriented computing & Web services

- **Service-oriented computing (SOC)**
  - an emerging paradigm for developing loosely coupled, interoperable, evolvable systems and applications

- **Web services** are a successful instantiation of SOC
Service-oriented computing & Web services

- **Service-oriented computing (SOC)**
  - an emerging paradigm for developing loosely coupled, interoperable, evolvable systems and applications

- **Web services** are a successful instantiation of SOC

```
WSDL
op.1 : type1
op.2 : type2
... op.n : typen
```

---

Scenario 4
A business process acts as an orchestrator, i.e. an active entity that invokes available services according to a given set of rules to meet a business requirement.

A business process is a service in its own right (compositionality).

Business processes may be composed of services of different granularities.
A business process acts as an *orchestrator*, i.e. an active entity that invokes available services according to a given set of rules to meet a business requirement.
Services & Business processes

- A *business process* acts as an *orchestrator*, i.e. an active entity that invokes available services according to a given set of rules to meet a business requirement.

A business process is a service in its own right (*compositionality*).

Business processes may be composed of services of different granularities.
Services & Business processes

- A *business process* acts as an *orchestrator*, i.e. an active entity that invokes available services according to a given set of rules to meet a business requirement.

A business process is a service in its own right (*compositionality*)

Business processes may be composed of services of different granularities
A business process acts as an orchestrator, i.e. an active entity that invokes available services according to a given set of rules to meet a business requirement.

A business process is a service in its own right (compositionality)

Business processes may be composed of services of different granularities.
Motivation

Goal
Developing formal reasoning mechanisms and analytical tools for checking that a service composition meets desirable correctness properties and does not manifest unexpected behaviors.

Approach: use Process Calculi
Being defined algebraically, they are inherently compositional thus convey in a distilled form the paradigm at the heart of SOC

- Provide linguistic formalisms for description of service-based applications and their composition
- Hand down a large set of reasoning mechanisms and analytical tools, e.g. typing systems and behavioural equivalences
Motivation

Goal
Developing formal reasoning mechanisms and analytical tools for checking that a service composition meets desirable correctness properties and does not manifest unexpected behaviors.

Approach: use Process Calculi
Being defined algebraically, they are inherently compositional thus convey in a distilled form the paradigm at the heart of SOC

- Provide linguistic formalisms for description of service-based applications and their composition
- Hand down a large set of reasoning mechanisms and analytical tools, e.g. typing systems and behavioural equivalences
COWS: a Calculus for Orchestration of Web Services

- Provide a linguistic formalism for description of service-based applications and their composition
- Model different (static and dynamic) aspects of SOC applications, e.g. publication, discovery, negotiation, orchestration, deployment, reconfiguration and execution
- Main features: service instantiation, correlation-based communication, forced termination, compensating activities, shared state, parallel with priority
COWS: a Calculus for Orchestration of Web Services

- Provide a linguistic formalism for description of service-based applications and their composition
- Model different (static and dynamic) aspects of SOC applications, e.g. publication, discovery, negotiation, orchestration, deployment, reconfiguration and execution
- Main features: service instantiation, correlation-based communication, forced termination, compensating activities, shared state, parallel with priority
Provide a linguistic formalism for description of service-based applications and their composition

Model different (static and dynamic) aspects of SOC applications, e.g. publication, discovery, negotiation, orchestration, deployment, reconfiguration and execution

Main features: service instantiation, correlation-based communication, forced termination, compensating activities, shared state, parallel with priority
Example: a translation (web) service
Example: a translation (web) service
Example: a translation (web) service

client service

resp

“albero”

req

word

Italian-English translation service
Example: a translation (web) service

client
service

resp

“tree”

req

word

Italian-English
translation
service

req

word
Example: a translation (web) service

client service

resp

"albero"

Italian-English translation service

req

word
Example: a translation (web) service
Example: a translation (web) service
Example: a translation (web) service

- A COWS high-level specification

\[ [x] \ t \cdot \text{req?}x \ . \ [y] \ t \cdot \text{word?}y \ . \ x \cdot \text{resp!}trans(y) \]
Example: a translation (web) service

- A COWS high-level specification

variable declarations

\[ [x] \text{ t \cdot req?x \cdot [y] t \cdot word?y \cdot x \cdot resp!trans(y)} \]
Example: a translation (web) service

- A COWS high-level specification

\[ [x] t \cdot \text{req?x} . [y] t \cdot \text{word?y} . x \cdot \text{resp!trans(y)} \]

variable declarations

receive activities
Example: a translation (web) service

- A COWS high-level specification

\[ [x] \ t \cdot \ req?x \ . \ [y] \ t \cdot \ word?y \ . \ x \cdot \ resp!trans(y) \]
Example: a translation (web) service

- A COWS high-level specification

\[
[x] \text{t} \cdot \text{req?x} : [y] \text{t} \cdot \text{word?y} : x \cdot \text{resp!trans(y)}
\]

- Variable declarations
- Prefixing
- Receive activities
- Invoke activities
Example: a translation (web) service

- A COWS high-level specification
  \[ [x] \cdot t \cdot \text{req}?x . [y] \cdot t \cdot \text{word}?y . x \cdot \text{resp}!\text{trans}(y) \]

- A COWS low-level specification

  \[ [\text{reqDB1},\text{reqDB2},\text{respDB1},\text{respDB2}] \ (\text{Translator} \mid \text{DB1} \mid \text{DB2}) \]
Example: a translation (web) service

- **A COWS high-level specification**
  
  \[ [x] t \cdot \text{req}?x \cdot [y] t \cdot \text{word}?y \cdot x \cdot \text{resp!trans}(y) \]

- **A COWS low-level specification**
  
  \[ [\text{reqDB1,reqDB2,respDB1,respDB2}] \ (\text{Translator} \mid \text{DB1} \mid \text{DB2}) \]
Example: a translation (web) service

- A COWS high-level specification

\[ [x] \ t \ \cdot \ \text{req}?x \ . \ [y] \ t \ \cdot \ \text{word}?y \ . \ x \ \cdot \ \text{resp}\!\text{trans}(y) \]

- A COWS low-level specification

\[ [\text{reqDB1}, \text{reqDB2}, \text{respDB1}, \text{respDB2}] \ (\text{Translator} \mid \text{DB1} \mid \text{DB2}) \]

\[
\text{DB1} = \begin{align*}
\text{t} \cdot \text{reqDB1}?\text{“a”} \ . & \ . \ t \cdot \text{respDB1}!\text{“to”} \\
+ & \text{t} \cdot \text{reqDB1}?\text{“albero”} \ . & \ . \ t \cdot \text{respDB1}!\text{“tree”} \\
+ & \ . & \ . \ . \ + \text{t} \cdot \text{reqDB1}?\text{“zucca”} \ . & \ . \ t \cdot \text{respDB1}!\text{“pumpkin”}
\end{align*}
\]

\[
\text{performs a quick search in a small database of commonly used words}
\]

\[
\text{DB2} = \begin{align*}
[\text{z}] \ ( & \text{t} \cdot \text{reqDB2}?\text{“zuppo”} \ . \ & \ . \ t \cdot \text{respDB2}!\text{“soaked”})
\end{align*}
\]

\[
\text{performs a slow search in a big database corresponding to trans()}
\]
Example: a translation (web) service

- A COWS high-level specification
  
  \[ [x] \cdot t \cdot req?x \cdot [y] \cdot t \cdot word?y \cdot x \cdot resp!trans(y) \]

- A COWS low-level specification
  
  \[ [reqDB1,reqDB2,respDB1,respDB2] \cdot (Translator \mid DB1 \mid DB2) \]

\[ [x] \cdot t \cdot req?x \cdot [y] \cdot t \cdot word?y \cdot [k] \cdot \]
\[ ( \cdot t \cdot reqDB1!y \mid [x1] \cdot t \cdot respDB1?x1 \cdot ( kill(k) | \{ x \cdot resp!x1 \} ) ) \]
\[ | \cdot t \cdot reqDB2!y \mid [x2] \cdot t \cdot respDB2?x2 \cdot ( kill(k) | \{ x \cdot resp!x2 \} ) ) \]

performs the two initial receives, and invokes DB1 and DB2 concurrently; when one of them replies, immediately stops the other search
Outline

1. Scenario
2. COWS
3. Verification problems
4. Symbolic semantics
5. Conclusion
Verification problems

- Verification tools (e.g. equivalence and model checkers) do not work directly on syntactic specifications
  - they work on abstract representations of the behaviour of terms

- Using an inappropriate representation for value-passing languages can lead to unfeasible verifications
  - For example, using standard Labelled Transition Systems (LTSs) for the COWS term $[x] t \cdot req?x . [y] t \cdot word?y . x \cdot resp!trans(y)$
Verification problems

- Verification tools (e.g. equivalence and model checkers) do not work directly on syntactic specifications
  - they work on *abstract* representations of the behaviour of terms
- Using an inappropriate representation for value-passing languages can lead to unfeasible verifications
  - For example, using standard Labelled Transition Systems (LTSs) for the COWS term $[x]t\cdot\text{req?}x\cdot[y]t\cdot\text{word?}y\cdot x\cdot\text{resp!}trans(y)$
Verification problems

- Verification tools (e.g. equivalence and model checkers) do not work directly on syntactic specifications
  - they work on *abstract* representations of the behaviour of terms
- Using an inappropriate representation for value-passing languages can lead to unfeasible verifications
  - For example, using standard Labelled Transition Systems (LTSs) for the COWS term \([x] t \cdot \text{req}?x . [y] t \cdot \text{word}?y . x \cdot \text{resp}!trans(y)\)
Verification problems

- Verification tools (e.g. equivalence and model checkers) do not work directly on syntactic specifications
  - they work on abstract representations of the behaviour of terms
- Using an inappropriate representation for value-passing languages can lead to unfeasible verifications
  - For example, using standard Labelled Transition Systems (LTSs) for the COWS term $[x] t \cdot \text{req?}x . [y] t \cdot \text{word?}y . x \cdot \text{resp!} trans(y)$

```
 Verification problems
+ Verification tools (e.g. equivalence and model checkers) do not work directly on syntactic specifications
  + they work on abstract representations of the behaviour of terms
+ Using an inappropriate representation for value-passing languages can lead to unfeasible verifications
  + For example, using standard Labelled Transition Systems (LTSs) for the COWS term $[x] t \cdot \text{req?}x . [y] t \cdot \text{word?}y . x \cdot \text{resp!} trans(y)$

```
Outline

1. Scenario
2. COWS
3. Verification problems
4. Symbolic semantics
5. Conclusion
Symbolic semantics

- To tackle the problem of infinite branching in the representations of COWS terms, we have defined a *symbolic semantics* for COWS.

**The main idea**

- A receive activities $p \cdot o?x$ can perform the action $p \cdot o?x$, that denotes reception of the generic value $x$ along the endpoint $p \cdot o$.

- To take care of the *admittable values* for generic values, we define a *symbolic semantics* where the label on each transition has two components:
  - the *condition* that must hold for the transition to be enabled
  - the *action* of the transition
Symbolic semantics

- To tackle the problem of infinite branching in the representations of COWS terms, we have defined a *symbolic semantics* for COWS.

The main idea

- A receive activities $p \cdot o?x$ can perform the action $p \cdot o?x$, that denotes reception of the generic value $x$ along the endpoint $p \cdot o$.

- To take care of the *admittable values* for generic values, we define a *symbolic semantics* where the label on each transition has two components:
  - the *condition* that must hold for the transition to be enabled
  - the *action* of the transition
Symbolic semantics: examples

- Symbolic LTS for
  \([x] t \cdot \text{req}\, x . [y] t \cdot \text{word}\, y . x \cdot \text{resp}\, !\, \text{trans}(y)\)

\[\begin{align*}
  &\text{t} \cdot \text{req}\, x \\
  &\text{t} \cdot \text{word}\, y \\
  &z = \text{trans}(y) , \\
  &x \cdot \text{resp}\, !z
\end{align*}\]
Symbolic semantics: examples

Symbolic LTS for
[reqDB1, reqDB2, respDB1, respDB2] (Translator | DB1 | DB2)
Symbolic semantics: examples

- Symbolic LTS for
  \([\text{reqDB1}, \text{reqDB2}, \text{respDB1}, \text{respDB2}] (\text{Translator} | \text{DB1} | \text{DB2})\)
Outline

1. Scenario
2. COWS
3. Verification problems
4. Symbolic semantics
5. Conclusion
Concluding remarks

- A symbolic LTS associated to a COWS term conveys in a distilled form all the semantics information of the term’s behaviour.

- Besides receive transitions, symbolic LTSs take into account:
  - generation and exportation of fresh names
  - pattern-matching
  - expressions evaluation
  - priorities among conflicting receives

- Dealing at once with all these features makes the symbolic semantics more complex than that for more standard calculi.

- The symbolic operational semantics for COWS can pave the way for the development of efficient equivalence and model checkers.

On-going and future work

- Definition of a notion of bisimilarity for COWS terms and efficient symbolic characterisation.
- Development of a model checker based on the symbolic semantics.
Concluding remarks

- A symbolic LTS associated to a COWS term conveys in a distilled form all the semantics information of the term’s behaviour.

- Besides receive transitions, symbolic LTSs take into account:
  - generation and exportation of fresh names
  - pattern-matching
  - expressions evaluation
  - priorities among conflicting receives

- Dealing at once with all these features makes the symbolic semantics more complex than that for more standard calculi.

- The symbolic operational semantics for COWS can pave the way for the development of efficient equivalence and model checkers.

On-going and future work

- Definition of a notion of *bisimilarity* for COWS terms and efficient symbolic characterisation
- Development of a model checker based on the symbolic semantics
Thank you!

For further details visit http://rap.dsi.unifi.it/cows/