Monitoring Conversational Web Services

Domenico Bianculli\textsuperscript{1} \hspace{1cm} Carlo Ghezzi\textsuperscript{2}

\textsuperscript{1}Faculty of Informatics - University of Lugano, Switzerland
\textsuperscript{2}Department of Electronics and Information - Politecnico di Milano, Italy

September 3, 2007
IW-SOSWE 2007
Agenda

1 Motivations

2 Conversational Web Services

3 Algebraic specifications

4 Monitoring Conversational Web services

5 Conclusions
Motivations

- SOC has fostered a very dynamic architectural style:
Motivations

- SOC has fostered a very dynamic architectural style:
  - binding among components may change at run-time

Web services live in an open-world [BDG]

Open-world software requires continuous validation

- validation must extend from development time to run-time
- Monitoring becomes a necessary component of a run-time validation facility
SOC has fostered a very dynamic architectural style:

- binding among components may change at run-time
- new services are published in registries

Web services live in an open-world [BDG]

Open-world software requires continuous validation

- validation must extend from development time to run-time
- Monitoring becomes a necessary component of a run-time validation facility
Motivations

- SOC has fostered a very dynamic architectural style:
  - binding among components may change at run-time
  - new services are published in registries
  - previously available services may disappear or become unavailable
Motivations

- SOC has fostered a very dynamic architectural style:
  - binding among components may change at run-time
  - new services are published in registries
  - previously available services may disappear or become unavailable
  - distribute ownership of composite services
Motivations

■ SOC has fostered a very dynamic architectural style:
  ➤ binding among components may change at run-time
  ➤ new services are published in registries
  ➤ previously available services may disappear or become unavailable
  ➤ distribute ownership of composite services

■ Web services live in an open-world [BDG]
Motivations

- SOC has fostered a very dynamic architectural style:
  - binding among components may change at run-time
  - new services are published in registries
  - previously available services may disappear or become unavailable
  - distribute ownership of composite services

- Web services live in an open-world [BDG]
- Open-world software requires continuous validation
SOC has fostered a very dynamic architectural style:
- binding among components may change at run-time
- new services are published in registries
- previously available services may disappear or become unavailable
- distribute ownership of composite services

Web services live in an open-world [BDG]

Open-world software requires continuous validation
- validation must extend from development time to run time
Motivations

- SOC has fostered a very dynamic architectural style:
  - binding among components may change at run-time
  - new services are published in registries
  - previously available services may disappear or become unavailable
  - distribute ownership of composite services

- Web services live in an open-world [BDG]

- Open-world software requires continuous validation
  - validation must extend from development time to run time
  - **Monitoring** becomes a necessary component of a run-time validation facility
Monitoring web services consists of:

- collecting data about the monitored service
- checking data conformance wrt expected service behavior
Monitoring Web Services

- Monitoring web services consists of:
  - collecting data about the monitored service
  - checking data conformance wrt expected service behavior

- There is considerable research on WS monitoring
Monitoring Web Services

- Monitoring web services consists of:
  - collecting data about the monitored service
  - checking data conformance wrt expected service behavior

- There is considerable research on WS monitoring

- ... previous work in our group on
  - monitoring services specified by a contract
  - contract defined via pre- and post-conditions
Monitoring web services consists of:

- collecting data about the monitored service
- checking data conformance wrt expected service behavior

There is considerable research on WS monitoring

... previous work in our group on

- monitoring services specified by a contract
- contract defined via pre- and post-conditions

All existing work deals with stateless Web services
Monitoring Web Services

- Monitoring web services consists of:
  - collecting data about the monitored service
  - checking data conformance wrt expected service behavior

- There is considerable research on WS monitoring

- ... previous work in our group on
  - monitoring services specified by a contract
  - contract defined via pre- and post-conditions

- All existing work deals with stateless Web services
  - ... are the existing approaches suitable for stateful Web services?
  - no! for example, pre- and post-conditions are inadequate for them
State-aware Web services taxonomy

- stateless
- stateful
State-aware Web services taxonomy

- **Stateless**
  - a.k.a. table-driven
  - they provide a functional abstraction
  - e.g. mathematical functions, zip code lookup

- stateful
State-aware Web services taxonomy

- stateless

Stateful

- Global Shared Resource services
- Conversational services
State-aware Web services taxonomy

- stateless
- stateful

Global Shared Resource services

- a.k.a. public-data-driven
- the computation depends on a common resource, shared among all instances of the services
- e.g. multi-user database, weather forecast service
State-aware Web services taxonomy

- stateless
- stateful

Conversational services

- a.k.a. private-data-driven
- the local state of the service depends on the conversation with its client
- they provide a data abstraction
- e.g. state-based calculations, shopping cart
The Shopping Cart example

```
«interface»
ShoppingCart

- insert(i: Item): void
- delete(i: Item): void
- amount(): int
```
The Shopping Cart example

used in an advanced shopping session described as a BPEL workflow...
Our proposal

Goal

to provide a run-time checking of the conformance of conversational services with respect to their specification.

monitored services are the ones a composite service, described in BPEL, interacts with.

How

by using a monitor component inside a BPEL engine, instrumented using AOP techniques.

Assumptions

the local workflow of a BPEL process is correct.

interactions with the external world may cause anomalies.
Our proposal

Goal

- to provide a run-time checking of the conformance of conversational services with respect to their specification
- monitored services are the ones a composite service, described in BPEL, interacts with
Our proposal

Goal

- to provide a run-time checking of the conformance of conversational services with respect to their specification
- monitored services are the ones a composite service, described in BPEL, interacts with

How

- by using a monitor component inside a BPEL engine, instrumented using AOP techniques
Our proposal

Goal

- to provide a run-time checking of the conformance of conversational services with respect to their specification
- monitored services are the ones a composite service, described in BPEL, interacts with

How

- by using a monitor component inside a BPEL engine, instrumented using AOP techniques

Assumptions

- the local workflow of a BPEL process is correct
- interactions with the external world may cause anomalies
Algebraic specifications are a well-known approach to formally specify software.
Algebraic specifications are a well-known approach to formally specify software.

It’s a formalism founded on the concept of an algebra.
Algebraic Specifications

- Algebraic specifications are a well-known approach to formally specify software.
- It’s a formalism founded on the concept of an algebra:
  - an algebra is a set of data with operations on those data.
Algebraic specifications are a well-known approach to formally specify software.

It’s a formalism founded on the concept of an algebra:
- an algebra is a set of data with operations on those data
- an algebraic specification is the description of an algebra
Algebraic specifications are a well-know approach to formally specify software.

It’s a formalism founded on the concept of an algebra:
- an algebra is a set of data with operations on those data
- an algebraic specification is the description of an algebra

It consists of:

- Axioms on the equivalence of sequences of operations implicitly describe the state of an object.
Algebraic Specifications

- Algebraic specifications are a well-known approach to formally specify software.
- It’s a formalism founded on the concept of an algebra:
  - an algebra is a set of data with operations on those data
  - an algebraic specification is the description of an algebra
- It consists of:
  - a signature declaring the types of the algebraic operations
Algebraic specifications are a well-known approach to formally specify software.

It's a formalism founded on the concept of an algebra:
- An algebra is a set of data with operations on those data.
- An algebraic specification is the description of an algebra.

It consists of:
- A signature declaring the types of the algebraic operations.
- A set of logical formulae describing properties of these operations.

Axioms on the equivalence of sequences of operations implicitly describe the state of an object.
Algebraic specifications are a well-known approach to formally specify software.

It’s a formalism founded on the concept of an algebra:
- An algebra is a set of data with operations on those data.
- An algebraic specification is the description of an algebra.

It consists of:
- A signature declaring the types of the algebraic operations.
- A set of logical formulae describing properties of these operations.

Axioms on the equivalence of sequences of operations implicitly describe the state of an object.
Algebraic specifications are a well-known approach to formally specify software. It's a formalism founded on the concept of an algebra:

- An algebra is a set of data with operations on those data.
- An algebraic specification is the description of an algebra.

It consists of:

- A signature declaring the types of the algebraic operations.
- A set of logical formulae describing properties of these operations.

Axioms on the equivalence of sequences of operations implicitly describe the state of an object.

Algebraic specifications are suitable for specifying conversational Web services!
import IntSpec, ItemSpec;

sort cart;

constructors
create() → cart;
insert(cart, item) → cart;

observers
amount(cart) → int;

transformers
delete(cart, item) → cart;

axioms
forall c: cart, i, j: item

amount(create()) = 0;
amount(insert(c,i)) = amount(c) + price(i);
delete(create(),i) = create();
delete(insert(c,i),j) =
  if (i == j) c
  else insert(delete(c,j),i);
end
Algebraic specifications evaluation

- Algebraic specifications can be evaluated by symbolically executing them.
Algebraic specifications evaluation

- Algebraic specifications can be evaluated by symbolically executing them.
- Evaluation is formally defined by a Term Rewriting System.
Algebraic specifications can be evaluated by symbolically executing them.

Evaluation is formally defined by a Term Rewriting System.

An interpreter (Eureka, CafeOBJ, ... ) works by applying a sequence of rewriting rules to a term, till the term reduces to a constant.
Consider the term deriving from this sequence of operations:

- create a cart; 2) insert item I_1; 3) insert item I_2; 4) delete item I_1; 5) insert item I_3; 6) get the amount of the cart

The output from CafeOBJ is:

```
CART> reduce amount(insert(I_3, delete(I_1, insert(I_2, insert(I_1, create())))))
```

1) apply trial-rule: `ceq delete(X:Item, insert(Y:Item, C:Cart)) = insert(Y, delete(X, C)) if X =/= Y`

2) apply trial-rule: `ceq delete(X:Item, insert(Y:Item, C:Cart)) = C if X == Y`

3) rule: `eq amount insert(X:Item, C:Cart) = X.price + amount(C)`

4) rule: `eq amount insert(X:Item, C:Cart) = X.price + amount(C)`

5) rule: `eq amount(create()) = 0`

6) rule: `eq :BDEMOD : M:Nat + N:Nat = #! (+ m n)`
Consider the term deriving from this sequence of operations:

1) create a cart; 2) insert item $I_1$; 3) insert item $I_2$; 4) delete item $I_1$; 5) insert item $I_3$; 6) get the amount of the cart
Consider the term deriving from this sequence of operations:

- 1) create a cart; 2) insert item \( I_1 \); 3) insert item \( I_2 \); 4) delete item \( I_1 \); 5) insert item \( I_3 \); 6) get the amount of the cart

the output from CafeOBJ is

```
CART> reduce amount(insert(I3, delete(I1, insert(I2, insert(I1, create())))))
1>[1] apply trial-- rule: ceq delete(X:Item,insert(Y:Item,C:Cart)) =
    insert(Y,delete(X,C)) if X /= Y
1<[1] delete(I1,insert(I2,insert(I1,create()))) --> insert(I2, delete(I1, insert(I1, create())))
1>[2] apply trial-- rule: ceq delete(X:Item,insert(Y:Item,C:Cart)) = C if X == Y
1<[2] delete(I1,insert(I1,create())) --> create()
1>[3] rule: eq amount insert(X:Item,C:Cart) = X.price + amount(C)
1<[3] amount insert(I3,insert(I2,create())) --> 3 + amount(insert(2,create()))
1>[4] rule: eq amount insert(X:Item,C:Cart) = X.price + amount(C)
1<[4] amount(insert(I2,create())) --> 2 + amount(create())
1>[5] rule: eq amount(create()) = 0
1<[5] amount(create()) --> 0
1<[6] 2 + 0 --> 2
1<[7] 2 + 3 --> 5
CART>
```
Consider the term deriving from this sequence of operations:

1) create a cart; 2) insert item \( I_1 \); 3) insert item \( I_2 \); 4) delete item \( I_1 \); 5) insert item \( I_3 \); 6) get the amount of the cart

The output from CafeOBJ is

```
CART> reduce amount(insert(I3, delete(I1, insert(I2, insert(I1, create())))))
1>[1] apply trial-- rule: ceq delete(X:Item,insert(Y:Item,C:Cart)) =
    insert(Y,delete(X,C)) if X /= Y
1<[1] delete(I1,insert(I2,insert(I1,create()))) --> insert(I2, delete(I1, insert(I1, create())))
1>[2] apply trial-- rule: ceq delete(X:Item,insert(Y:Item,C:Cart)) = C if X == Y
1<[2] delete(I1,insert(I1,create())) --> create()
1>[3] rule: eq amount insert(X:Item,C:Cart) = X.price + amount(C)
1<[3] amount insert(I3,insert(I2,create())) --> 3 + amount(insert(2,create()))
1>[4] rule: eq amount insert(X:Item,C:Cart) = X.price + amount(C)
1<[4] amount(insert(I2,create())) --> 2 + amount(create())
1>[5] rule: eq amount(create()) = 0
1<[5] amount(create()) --> 0
1<[6] 2 + 0 --> 2
1<[7] 2 + 3 --> 5
CART>
```

time for the evaluation on a consumer laptop: 0.040s
The architecture adopts an aspect-oriented programming approach
The architecture adopts an aspect-oriented programming approach
  ▶ aspects are attached to a BPEL engine to monitor service compositions
The architecture adopts an aspect-oriented programming approach

- aspects are attached to a BPEL engine to monitor service compositions
- we use AspectJ and ActiveBPEL
The architecture adopts an aspect-oriented programming approach

- aspects are attached to a BPEL engine to monitor service compositions
- we use AspectJ and ActiveBPEL

AOP allows to separate business and monitoring logics
The architecture adopts an aspect-oriented programming approach
- aspects are attached to a BPEL engine to monitor service compositions
- we use AspectJ and ActiveBPEL

AOP allows to separate business and monitoring logics

Overhead due to monitoring instrumentation is limited (5% execution time)
Monitoring Architecture (2/3)

ActiveBPEL engine

AOP

Main Interceptor

Specifications registry

Monitor
The standard ActiveBPEL implementation is extended with three main components:

- Main Interceptor
The standard ActiveBPEL implementation is extended with three main components:

- **Main Interceptor**
  - it intercepts and modifies the execution of a process within the engine at some pointcuts
The standard ActiveBPEL implementation is extended with three main components:

- **Main Interceptor**
  - it intercepts and modifies the execution of a process within the engine at some pointcuts
  - pointcuts correspond to the execution of *invoke* activities
The standard ActiveBPEL implementation is extended with three main components:

- **Main Interceptor**
  - it intercepts and modifies the execution of a process within the engine at some pointcuts
  - pointcuts correspond to the execution of `invoke` activities

- **Specification registry**
The standard ActiveBPEL implementation is extended with three main components:

- **Main Interceptor**
  - it intercepts and modifies the execution of a process within the engine at some pointcuts
  - pointcuts correspond to the execution of *invoke* activities

- **Specification registry**
  - it contains the specifications against which services are checked for conformance
The standard ActiveBPEL implementation is extended with three main components:

- **Main Interceptor**
  - it intercepts and modifies the execution of a process within the engine at some pointcuts
  - pointcuts correspond to the execution of **invoke** activities

- **Specification registry**
  - it contains the specifications against which services are checked for conformance

- **Monitor**
The standard ActiveBPEL implementation is extended with three main components:

- **Main Interceptor**
  - it intercepts and modifies the execution of a process within the engine at some pointcuts
  - pointcuts correspond to the execution of `invoke` activities

- **Specification registry**
  - it contains the specifications against which services are checked for conformance

- **Monitor**
  - the actual conformance checker
Conformance checker integration

Interpreter (CafeOBJ, Heureka) \rightarrow \text{Monitor} \rightarrow \text{Symbolic state generator}

\text{evaluation} \rightarrow \text{state} \rightarrow \text{constructors transformers}

\text{observers}
Conformance checker integration

- the *symbolic state generator* keeps a machine-readable description of the state of a service
- the state is updated when a *constructor/transformer* is invoked
- a call to an *observer* triggers state evaluation by the interpreter
the interpreter returns a constant, resulting from the evaluation of the state

this value is then checked wrt the real value from the invocation of the monitored service

if there’s a mismatch, an user-defined action is performed
We propose a technique to monitor the functional behavior of conversational web services
Conclusions

- We propose a technique to monitor the functional behavior of conversational web services
  - We use algebraic specifications to specify services
Conclusions

- We propose a technique to monitor the functional behavior of conversational web services
  - We use algebraic specifications to specify services
  - We embed the monitor inside a BPEL engine using AOP
Conclusions

- We propose a technique to monitor the functional behavior of conversational web services
  - We use algebraic specifications to specify services
  - We embed the monitor inside a BPEL engine using AOP
- Future work
Conclusions

- We propose a technique to monitor the functional behavior of conversational web services
  - We use algebraic specifications to specify services
  - We embed the monitor inside a BPEL engine using AOP

- Future work
  - Experimenting with and evaluating different interpreters of algebraic specifications
Conclusions

- We propose a technique to monitor the functional behavior of conversational web services
  - We use algebraic specifications to specify services
  - We embed the monitor inside a BPEL engine using AOP

- Future work
  - experimenting with and evaluating different interpreters of algebraic specifications
  - integrating this work within a comprehensive environment for design- and run-time validation of Web service compositions