
Web Services Composition

Nikola Milanovic
Humboldt University, Berlin
milanovi@informatik.hu-berlin.de

Table of Contents

- Service Oriented Computing
- Existing proposals for service composition (BPEL4WS)
- Alternative solutions
- Conclusion

Service Oriented Computing

- Computing paradigm that uses services as basic building blocks for application development
- Services:
 - Self-describing, open components
 - Rapid deployment, reuse, interoperability
- Native capabilities of SOC:
 - Publication
 - Discovery
 - Selection
 - Binding

Service Composition

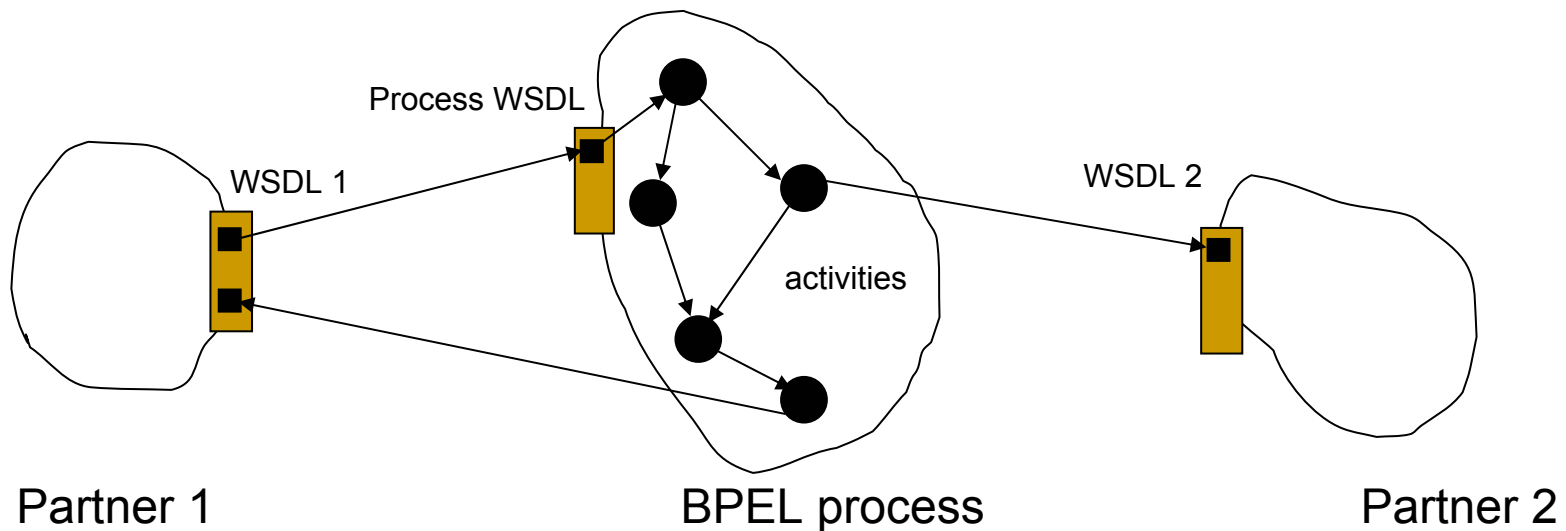
- Layered Service Oriented Architecture:
 - Native capabilities:
 - Publication
 - Discovery
 - Selection
 - Binding
 - Composition
 - Coordination (flow)
 - Conformance (integrity)
 - Monitoring (events)
 - QoS
 - Management

Existing Solutions – BPEL4WS

- BPEL4WS = Business Process Execution Language for Web Services
- XML based language for description of Web service choreographies according to the specified business rule – business workflows
- BPEL offers:
 - Definition of business protocols
 - Fault handling and compensation

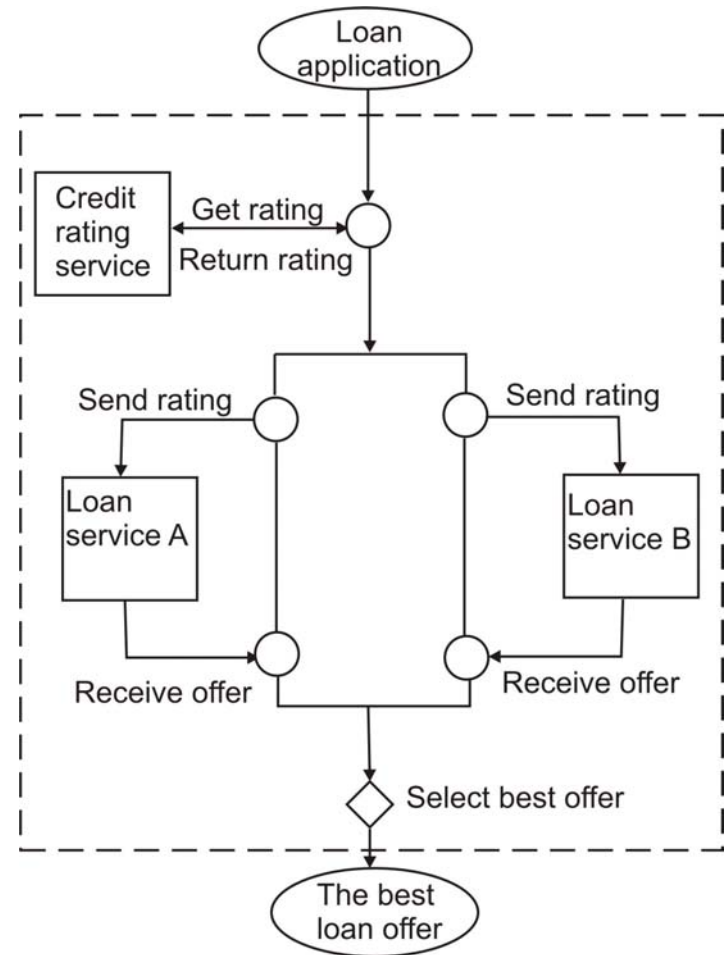
BPEL Terminology

- Process (BPEL composition)
- Partners (participating services)
- Activities (message exchange)



BPEL Example

- Process is defined:
 - BPEL4WS source file (.bpel)
 - Process interface (.wsdl)
 - Deployment descriptor (.xml)



BPEL Syntax

- **Basic XML elements describing BPEL flow:**
 - `<process>`
 - `<invoke>`, `<invoke>...<receive>`
 - `<variable>`, `<assign>`, `<copy>`
 - `<scope>`, `<faultHandlers>`
 - `<flow>`, `<sequence>`
 - `<switch>`

Anatomy of a BPEL Process

<code><process... /></code>	process declaration
<code><partnerLinks... /></code>	partner services
<code><variables... /></code>	in, out and temp vars
<code><sequence></code>	
<code><receive... /></code>	start process
<code><assign><copy.../></assign></code>	store, copy values
<code><sequence></code>	
<code><invoke... /><assign><copy.../></assign></code>	invoke sync service
<code></sequence></code>	
<code><flow></code>	
<code><sequence></code>	invoke 2 async services
<code><invoke><receive.../></invoke></code>	in parallel
<code></sequence></code>	
<code><sequence></code>	
<code><invoke><receive.../></invoke></code>	
<code></sequence></code>	
<code></flow></code>	
<code><switch></code>	decide
<code></sequence></code>	
<code></process></code>	end process

Problems of BPEL

- Does not address conformance and QoS
- Deals with connectivity only, not with correctness
- Turing-complete language, more about implementation than specification
- Cannot verify properties of a composition result

Alternative Solutions to Web Services Composition

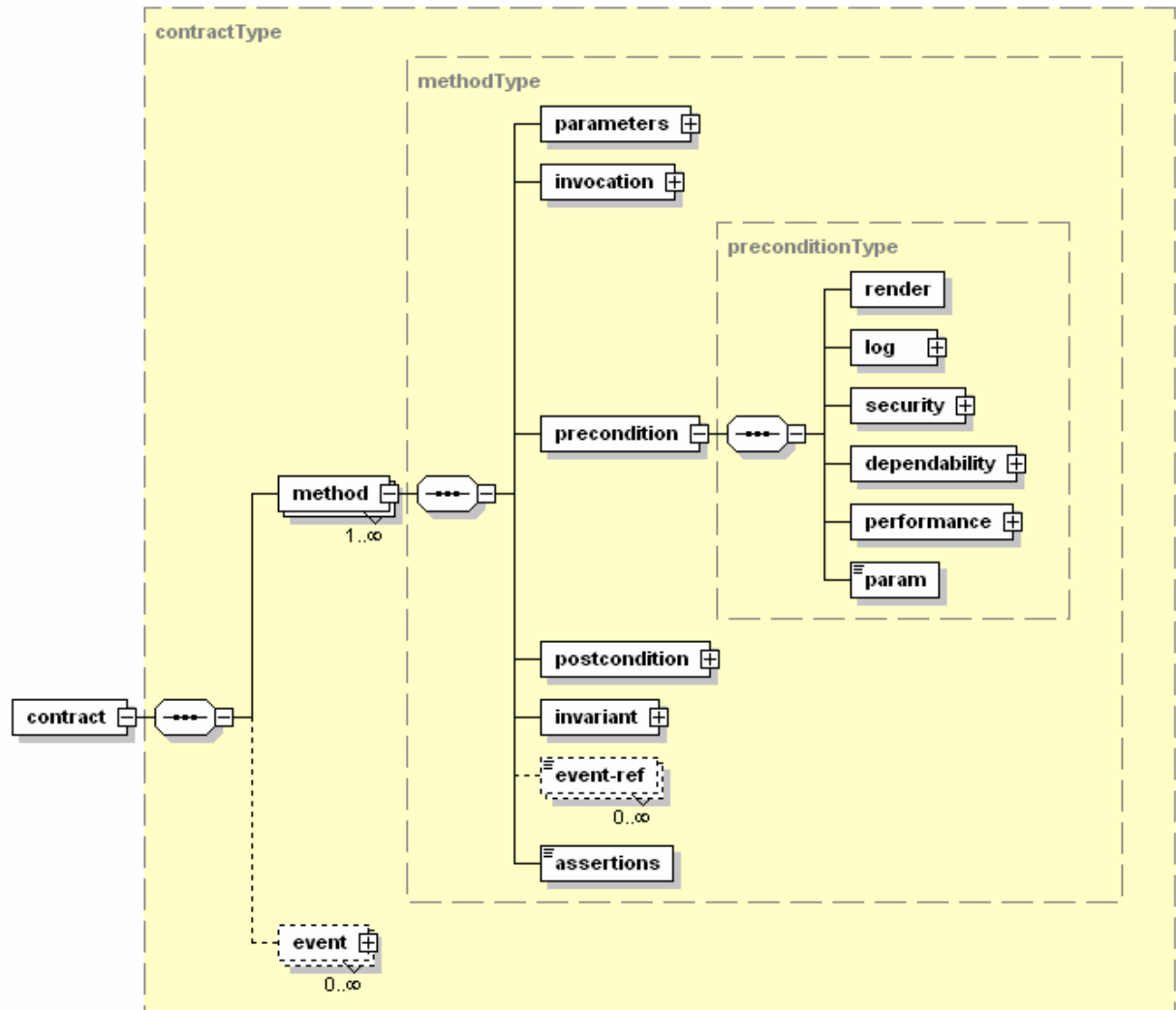
- Web Component
 - Dual service definition (XML/class)
 - Reuse, specialization, extension
- π - Calculus
 - Port (name), message
 - Adding contracts to ports
- Abstract machines

Modeling Services as Abstract Machines

- Service contracts
- Directories
- Formal service description
 - Predictability
 - Correctness
 - Proof obligation
- Composable architecture
 - Behavior
 - Composition operators

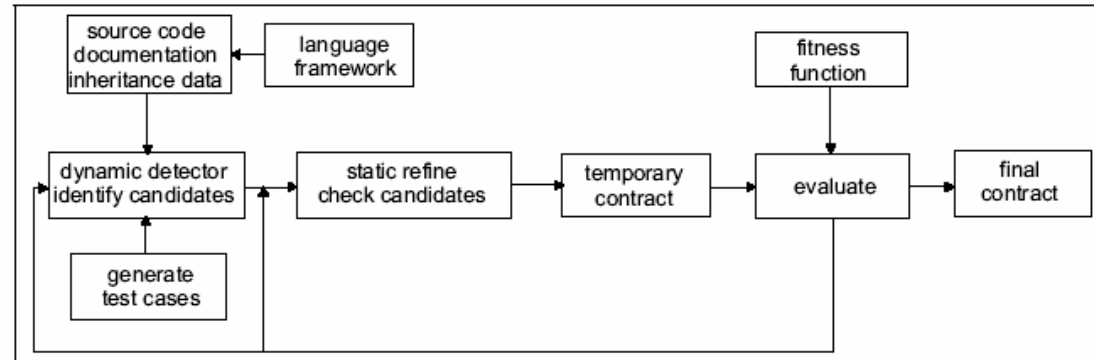
Contracts for Web Services

- Extends/
supplements
WSDL:
 - Non-functional
properties
 - Preconditions,
postconditions,
invariants
 - Events



Generating, Storing and Processing Service Contracts

- Contract Definition Language (CDL)
- Service directories
- Extracting contracts
 - A priori extraction
 - A posteriori extraction



Formalized Contracts

- Transferring CDL into formal notation
- Modeling service contract as an abstract machine
 - Statics (state variables)
 - Dynamics (functions)

```
MACHINE M(X, x)
CONSTRAINTS C
CONSTANTS Ct
SETS S
PROPERTIES P
VARIABLES V
INVARIANT I
ASSERTIONS J
INITIALIZATION U
OPERATIONS u ← O(w) PRE Q THEN V END
END
```

Composable Service Architecture

- Architecture: set of initial services and composition operators
 - State variables are assigned behavior:
 - Invariant
 - Bounded
 - Vanishing
 - Emerging
 - Transferred
 - Static and dynamic behavior assignment
-

Composition Operators

- Behavior table B:

$$B[i,j]=operation(E_i,C_j)$$

$$operation: E \times C \rightarrow O$$

$E = \{\text{parameters, invocation, semantic...}\}$

$C = \{\text{invariant, bounded, vanishing...}\}$

$O = \{+, -, \cup, /, \cap \dots\}$

- Dynamic behavior assignment:

$$\forall e \in E, \forall c \in C (\exists! o \in O \mid behavior(e,c) = o)$$

Proof Obligation

- Composition is achieved by merging two abstract machines using composition operators
- Proof obligation
 - Initialization:
 $C \wedge P \Rightarrow [U]I$
 - Assertion:
 $C \wedge P \wedge J \Rightarrow I$
 - Operation:
 $C \wedge P \wedge J \wedge Q \Rightarrow [V]I$
- Composition operators enable **predictability**, proof obligation enables **correctness**.

Example of Merging Two Abstract Machines

```

MACHINE creditRating
VARIABLES loanApplication(IN):invariant; creditRating(OUT):vanish
INVARIANT loanApplication ∈ Application:invariant
OPERATIONS rating(SYNC) ≜ PRE THEN creditRating=rating(loanApplication):transfer;
wcet=300ms:vanish; ncc=10:bounded
EXCEPTIONS invalidApplication:transfer
    
```

	exception	operation	ncc
transfer	∪	SEQUENCE	+
bound	/	/	min
NO_MATCH	/	/	/

```

MACHINE loanCompany
VARIABLES creditRating(IN):vanish loanOffer(OUT):invariant interestRate(OUT):invariant
INVARIANT loanOffer ≥ 0
OPERATIONS offer(ASYNC) ≜ PRE THEN loanOffer=offer(creditRating):transfer;wcet=24h:vanish;
ncc=1000:bounded;
EXCEPTIONS invalidRating:transfer noLoan:invariant
    
```

newService = creditRating x
switch(loanCompanyA,loanCompanyB)_{min(interestRate)}

```

MACHINE newService
VARIABLES loanApplication(IN):invariant loanOffer(OUT): invariant
INVARIANT loanApplication ∈ Application loanOffer ≥ 0
OPERATIONS bestLoan (ASYNC) ≜ PRE THEN loanOffer=bestLoan(loanApplication):transfer;ncc=10
EXCEPTIONS invalidAppRating:transfer noLoan:invariant
    
```

A Look Ahead

- Automatic service composition:
 - Specifying target contracts
 - Human intervention minimized
 - Separates “what” from “how”

Conclusion

- After connectivity between services is established, two key properties of the composition are:
 - Predictability
 - Correctness
- Extremes:
 - Pure connectivity (WSDL)
 - Pure implementation (BPEL)
- We need a model capable of specifying behavior of services and automatic verification of crucial properties of their composition