A Dynamic and Reactive Approach to the Supervision of BPEL Processes

L. Baresi, S. Guinea
[baresi|guinea]@elet.polimi.it
Outline

- The scenario - *the challenge*
- A case study - *medical tele-assistance*
- Supervised Processes - *what we provide*
- WSCoL/WSReL
- The Dynamo Framework
- Future Work
The challenge

- **Composition**: build intrinsically distributed and dynamic systems by leveraging remote services

  Dynamism/Flexibility/Distributed Ownership/Open World

  pre-deployment validation is only part of the solution!

  (assume/guarantee)

  unexpected and catastrophic events can arise!

- **Self-Healing Processes (monitoring/recovery)** [1]
  - detect faults/errors instantly
  - contain the effects
  - recover and proceed if possible

Supervised processes

- A holistic approach that does NOT only consider the process but also:
  - who is running it
  - when it is being run
  - the environment in which it is being run
Supervised processes

- A holistic approach that does NOT only consider the process but also:
  - who is running it
  - when it is being run
  - the environment in which it is being run

- Ways to complete the process design with supervision activities
  - a declarative indication of the functionalities and QoS that should be guaranteed at run time
    - pre-/post-conditions on the interactions the process has with the outside world - specified using WSCoL (*Web Service Constraint Language*).
  - an indication of the recovery strategies to be used to keep things on track
    - a set of strategies built from a set of atomic recovery actions we provide to the designer - specified using WSReL (*Web Service Recovery Language*).
Supervised processes

- A holistic approach that does NOT only consider the process but also:
  - who is running it
  - when it is being run
  - the environment in which it is being run

- Ways to complete the process design with supervision activities
  - a declarative indication of the functionalities and QoS that should be guaranteed at run time
    - pre-/post-conditions on the interactions the process has with the outside world - specified using WSCoL (Web Service Constraint Language)
  - an indication of the recovery strategies to be used to keep things on track
    - a set of strategies built from a set of atomic recovery actions we provide to the designer - specified using WSReL (Web Service Recovery Language)

- A framework (Dynamo) that augments a BPEL engine with self-healing capabilities and allows for separation of concerns
Location: where in the process the rule will be considered

Supervision Parameters: meta-level information for switching supervision activities on and off

Monitoring Property: declarative specification of what should be true in that location

Recovery strategy: conditional branching allows us to choose the most fitting strategy
WSCoL/WSReL

- **WSCoL**
  - Mixes JML and XML technologies
  - Characteristics:
    - internal/external/historical variables and aliasing
    - boolean/relational/mathematical operators
    - universal/existential quantifiers
    - aggregate functions
    - data type related functions
WSReL

- instance recovery
- modifies how the process will play out from that point on (no backward recovery)
- Allows the definition of multiple alternative recovery strategies
  - which is chosen depends on a WSCoL condition
- Each strategy is made up of recovery steps
  - take a step and check if the problem is fixed, if not proceed to next step
- Each step is made up of atomic recovery actions:
  - ignore/notify/halt/retry/rebind/change_sup_rules/change_sup_params/change_process_params/substitute/call/callback

```java
if (condition1) {
    step 1 ||
    step2
}
else (condition2) {
    step1
}
```
We want to supervise how the drug dosage changes

**location**: [invoke] analyzeData/post-condition

**parameters**: priority = 1

**monitoring property**:

\[
\text{let } \$doseNew = (\text{labResults/suggestedDose}); \\
\text{let } \$doseOld = \text{retrieve(pID, uID, iID,’[INVOKE]changeDose/post-condition’,’doseStored’, 1);} \\
\$doseNew \leq \$doseOld \times 1.05 \text{ and } \$doseNew \geq \$doseOld \times 0.95;
\]

**recovery strategy**:

\[
\text{if } (\$doseNew \leq \$doseOld \times 1.075 \text{ and } \$doseNew \geq \$doseOld \times 0.925) \{ \\
\text{retry}(1) \text{ or} \\
\text{notify (mess, admin_email)} \text{ and rebind (second_LAB_URI)} \text{ or} \\
\text{notify (mess, admin_email)} \text{ and substitute (private_LAB, ‘analyze’, data, XSLT)} \text{ or} \\
\text{callback (‘Emergency’, input)}
\}
\text{else } \\
\text{notify (mess, admin_email)} \text{ and callback (‘Emergency’, input)}
\]
Dynamo - General Architecture

Execution Engine

- ActiveBPEL + AspectJ

Configuration Manager
- Java Persistence

Recovery Manager
- Java

Supervision Manager
- Java/Antr

WSCoL Analyzer
- Java/Antr

Dynamic Invoker
- jax-ws

Storage
- Java Persistence

Execution and Supervision Environment

Java Persistence

Java/Antr

jax-ws
Self-Healing Capabilities are added to the ActiveBPEL Execution Engine using AspectJ

Main run-time components:
- Supervision Manager
  - collects the internal/external/historical variables
  - gets the properties to be checked and the strategies to be executed
- WSCoL Analyzer
  - checks the properties
- Recovery Manager
  - executes the atomic actions in a recovery step
Self-Healing Capabilities are added to the ActiveBPEL Execution Engine using AspectJ

Main run-time components:
- **Supervision Manager**
  - collects the internal/external/historical variables
  - gets the properties to be checked and the strategies to be executed
- **WSCoL Analyzer**
  - checks the properties
- **Recovery Manager**
  - executes the atomic actions in a recovery step
Self-Healing Capabilities are added to the ActiveBPEL Execution Engine using AspectJ.

Main run-time components:

- **Supervision Manager**
  - collects the internal/external/historical variables
  - gets the properties to be checked and the strategies to be executed

- **WSCoL Analyzer**
  - checks the properties

- **Recovery Manager**
  - executes the atomic actions in a recovery step
Self-Healing Capabilities are added to the ActiveBPEL Execution Engine using AspectJ

Main run-time components:

- **Supervision Manager**
  - collects the internal/external/historical variables
  - gets the properties to be checked and the strategies to be executed

- **WSCoL Analyzer**
  - checks the properties

- **Recovery Manager**
  - executes the atomic actions in a recovery step
Self-Healing Capabilities are added to the ActiveBPEL Execution Engine using AspectJ

Main run-time components:

- **Supervision Manager**
  - collects the internal/external/historical variables
  - gets the properties to be checked and the strategies to be executed

- **WSCoL Analyzer**
  - checks the properties

- **Recovery Manager**
  - executes the atomic actions in a recovery step
Self-Healing Capabilities are added to the ActiveBPEL Execution Engine using AspectJ

Main run-time components:

- **Supervision Manager**
  - collects the internal/external/historical variables
  - gets the properties to be checked and the strategies to be executed

- **WSCoL Analyzer**
  - checks the properties

- **Recovery Manager**
  - executes the atomic actions in a recovery step
Recovery Manager

```java
public boolean recover() {
   /*
   1. while stack contains elements {
      1.1 pop 1 element from stack
      1.2 read element
      1.3 If element is analyze (cond, n)
         1.3.1 analyze(cond, n)
      1.4 else
         1.4.1 execute recovery step
         1.4.2 check step through monitoring
         1.4.3 if monitoring result is true
            1.4.3.1 return true
         1.4.4 if monitoring result is false
            1.4.4.1 rollback step
   }
   2. return false
   */
}

private void analyze(String condition, int numSteps) {
   /*
   1. if condition == null
      1.1 return
   2. send condition and monitoring data to WSCoL analyzer.
   3. if analyzer answers true
      3.1 return
   4. if analyzer answers false
      4.1 pop numSteps element from stack
      4.2 return
   */
}
```

```
// stack
analyze(condition1, 4)
retry(1)
notify(mess, admin_email) &&
rebind(second_LAB_URI)
notify(mess, admin_email) &&
call(private_LAB, 'analyze', data, XSLT)
notify(mess, admin_email) &&
callback('Emergency', input)
analyze(null, 1)
notify (mess, admin_email) &&
callback('Emergency', input)
```
Future Work

- Asynchronous approach with temporal properties - *some work has been done*
- Take advantage of level of technical expertise to improve recovery approach
  - AOP gives us access to the run-time rep of the process
    - backward recovery vs. forward recovery - *some work has been done*
    - process re-organization
    - monitoring may also lead to changes in the process definition and to changes in other processes
- Take advantage of the intrinsic distributed nature of web services
  - try to move towards decentralized supervision techniques - *some work has been done*
Supervision Parameters

- **Priority levels:**
  - each supervision rule has a priority level (integer number)
  - each running process instance indicates a threshold priority level
  - if the rule’s priority is less or equal to the process’, the rule is considered
  - Changing the process’ priority level is like turning a knob!

- **Temporal validity:**
  - indicates a time-frame in which the rule must be checked (*from/to*) [2], or
  - indicates max number of times it must be checked (*X times*).

- **Delay:**
  - indicates minimum delay between two supervisions (*wait for 3d12h*).
  - indicates the number of skips between two supervisions (*wait for 3X*).

- **Trusted providers:**
  - indicates a list of providers for which supervision is always unnecessary
  - can be useful when the system uses dynamic binding

---

[2] ISO 8601 string format for complete dates containing hours, minutes, and seconds