Formal Modelling and Verification of an Asynchronous Extension of SOAP

Maurice ter Beek
FM&&T, ISTI–CNR, Pisa, Italy

Wednesday 6 December
ECOWS 2006

joint work with: Stefania Gnesi and Franco Mazzanti (FM&&T, ISTI–CNR)
Corrado Moiso (Telecom Italia)
Outline

- Context: EU project Sensoria
- Why extend SOAP to aSOAP?
- Design and implementations of aSOAP
- Model checking with UMC and $\mu$-UCTL
- Conclusions and future work
Why aSOAP?

Current WS largely use SOAP on top of HTTP:

- blocks Clients between request and response
- request fails if no response within (limited) period of time

Next-generation telecom networks need asynchronous extension of SOAP for asynchronous interactions among distributed WS, to deal with:

- long-running computations
- temporary unavailability of WS consumer or WS provider

⇒ ongoing convergence of telecom & Internet worlds into a single application context requires modern WS to integrate telco features
An approach to handle asynchronous MEPs in the context of WS

Other approaches include ASAP, WSN, WS-Reliable Messaging and WS-Reliability by OASIS and Pubscribe by Apache:

+ ASAP can handle long-running computations
  - ASAP results returned as untyped XML “blobs”
  - no direct support for asynchronous two-way MEPs
  - no direct support for temporary Client/Server unavailability
  - all operate on application level

⇒ aSOAP allows asynchronous interaction on protocol level by extending SOAP
Assumptions

- Proxy is always reachable by both Client and Server whenever they have an active connection.
- If Client is willing to accept an asynchronous response to SOAP Invocation, then it inserts the URL of the SOAP listener where it wants to receive the response in the SOAP header.
- The URL in the header of SOAP Invocation (URL) is the address of a generic SOAP listener; the application level has a mechanism to receive SOAP messages at this URL.
- Upon receiving an asynchronous SOAP Invocation (URL) from Client, Proxy generates REQ-ID to uniquely identify Client’s SOAP Invocation in further communications.
aSOAP scenario

HTTP-REQ(SOAP Invocation(URL))

HTTP-REQ(SOAP Invocation(REQ-ID))

HTTP-REQ(SOAP Invocation(REQ-ID))

HTTP-REQ(SOAP Invocation(REQ-ID))

HTTP-RES(SOAP Unreachable(REQ-ID))

HTTP-REQ(SOAP Invocation(URL))

Proxy tries to contact the Server until Server becomes reachable or retry policy times out

Client becomes unreachable

Server becomes unreachable

HTTP-RES(SOAP Deferred)

Server elaborates response

HTTP-REQ(SOAP Result(REQ-ID))

HTTP-RES(SOAP OK)

Proxy tries to contact the Client until Client becomes reachable or retry policy times out

HTTP-REQ(SOAP Result(REQ-ID))

HTTP-REQ(SOAP Result(REQ-ID))

HTTP-REQ(SOAP Result(REQ-ID))

HTTP-REQ(SOAP Result(REQ-ID))

HTTP-RES(SOAP OK)
Design and implementation

aSOAP is designed to have *minimal impact* on existing architectures:
- preserves *backward compatibility* with SOAP
- Client without asynchronous invocations *need not be modified*
- concentrates *overhead* resulting from extension in Proxy

aSOAP can be implemented by using SOAP v1.2:
- permits addition of SOAP headers (URL) to SOAP messages
- aSOAP headers can thus be SOAP v1.2 headers
- Proxy can be intermediary node processing aSOAP headers
We present a first step in the development of aSOAP:

- use formal methods to analyse an initial formalisation
- use conclusions to eventually arrive at a formal proposal

Methodology:

- specify aSOAP as a set of communicating UML state machines
- resulting semantics are doubly-labelled transition systems
- express behavioural properties in the action- and state-based temporal logic $\mu$-UCTL
- verify with on-the-fly model checker UMC (developed by FM&&T, ISTI–CNR)
**Model checking**

Automatic technique to verify whether a (concurrent) system design satisfies its specifications

Very hard in standard ways (like testing) due to non-determinism and interleaving

+ exhaustive verification: takes into account all possible input combinations and states

− state-space explosion: risk of running out of memory

⇒ use a simplified model, still capturing core of system design, while abstracting from unnecessary details
Only asynchronous SOAP invocations: no backward compatibility

URL in header of SOAP message is identified with Client: each Client is just a listener of asynchronous SOAP invocations

System model: 1 Server (+ subthreads), 1 Proxy (+ subthreads) and fixed (configurable) # Clients

Proxy and Server may activate at most a fixed (configurable) # parallel subthreads

Client or Server unreachable: Proxy attempts to contact them at most a fixed (configurable) # times

Client issues a single SOAP invocation and then terminates

Future work: Client performs loop of SOAP invocations or issues several SOAP invocations before waiting for deferred SOAP results
Excerpt from Client specification

Class Client is
Operations:
    SOAP_Result(requid:Tokens):Tokens; SOAP_Failure(requid:Tokens):Tokens;
Vars:
    status: Tokens := Inactive; theproxy: Proxy; result: Tokens[] := [];
State top = ready, check, wait, done
Transitions:
    ready -> check // Proxy always reachable
        { - / status := Running; result := theproxy.PSOAP_Invocation(self) } 
    check -> wait // Server initially unreachable
        { -[result[0]=Server_Unreachable] / result := []; } // await results
    check -> wait // Connection with Proxy lost
        { -[result[0]=Client_Unreachable] / result := []; } // await results
    check -> done // Immediate result from Server
        { -[result[0]=Soap_Result] / status := Done; result := []; } 
    wait -> done // Issued invocation to Proxy
        { SOAP_Result(requid) / status := Done; // await results from Proxy to return Soap_OK; } // complete listener execution
...
end Client;
**Statechart specification of Client**

```
ready

- / status := Running; result :=
  theproxy.PSOAP_Invocation(self)

check

- [result[0]=Server.Unreachable] / result := []
- [result[0]=Client.Unreachable] / result := []
- [result[0]=Soap.Deferred] / result := []

wait

- [result[0]=Server.Unreachable] / result := []

done

- [result[0]=Server.Result] /
  status := Done; result := []

SOAP_Failure(reqid) /
  status := Done;
  return(Soap_OK)

SOAP_Result(reqid) /
  status := Done;
  return(Soap_OK)
```
# State-space complexity

<table>
<thead>
<tr>
<th>Clients</th>
<th>attempts</th>
<th>P subthreads</th>
<th>S subthreads</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>151</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>319</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>151</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>319</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>151</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>319</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3701</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8321</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>23953</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>97699</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>87569</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>393907</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>67099</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>151414</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>&gt; 500000</td>
</tr>
</tbody>
</table>


Property: *all system executions eventually reach a configuration in which all Clients are in status* Done

In $\mu$-UCTL:

$$AF ((C1.status=Done) \land (C2.status=Done))$$

⇒ False: Server’s response need not reach Client

Not surprising, possible scenario: Client’s SOAP invocation is deferred by Server, but its subsequent final SOAP result never reaches Client (because Client is unreachable for a sufficiently long time for Proxy to cancel the SOAP invocation)
Client and Server always reachable?

Property: *for all paths without communication failures the system will eventually reach a configuration in which all Clients are in status Done*

In $\mu$-UCTL’s **minimal fixed point** structure:

\[
\min Z: \\
((C1.status=Done) \text{ and } (C2.status=Done)) \\
or((PT1.result=Client\_Unreachable) \\
or(PT1.result=Server\_Unreachable) \\
or(PT2.result=Client\_Unreachable) \\
or(PT2.result=Server\_Unreachable)) \\
or(\text{not FINAL} \text{ and } [true] Z)
\]

⇒ True: UMC analysed 34735 states
Unexpected result

Property: if Client receives SOAP_Result(ReqId) operation call, then it received [Soap_Deferred, ReqId] response to its previous SOAP_Invocation

In 𝜇-UCTL:

$$\text{AG} \ [\text{C1.SOAP}_\text{Result}(\ast,\text{ReqId})]$$

$$(\text{C1.result}=[\text{Soap}_\text{Deferred}, \text{ReqId}])$$

⇒ Formula should obviously be false, but unfortunately it is true

Possible reason: Proxy finds Client unreachable and is unable to notify Client of deferred Server response + REQ-ID (this of course does not prevent request to proceed as usual, until eventually the deferred result is produced by Server, but REQ-ID will mean nothing to Client)

⇒ Future work: gravity of this particularity and a way to avoid it
Conclusions and future work

- Ongoing work: applying academic experience with formal modelling and verification to an industrial case study
  - ISTI–CNR & Telecom Italia: partners in EU project Sensoria on Software Engineering for Service-Oriented Overlay Computers
  - Goal: use formal methods in design phase of aSOAP to arrive at a formal proposal of which we can guarantee certain properties
  - Future work: use UMC for quite bigger specifications and more intriguing properties