Service Choreographies: The Case for Interaction Paradigm Interoperability

Nikolaos Georgantas, Georgios Bouloukakis, Valérie Issarny

MiMove team, Inria Paris-Rocquencourt

Journée AFSEC, Paris, 16/10/14
Service choreography: Search & Rescue (S&R)

- Life sign sensors
- S&R personnel
- Info diffusion & coordination

Middleware interoperability required

S&R service choreography

JMS event-based system

JMEDS Web service

Planning & deployment

S&R coordinator

Monitoring & control centers

Web Services
JavaSpaces
JMS

Heterogeneous middleware interaction protocols
Interaction paradigms in middleware

- Look for comprehensive systematic solution to middleware interoperability
- Classify middleware implementations into families
- Families follow well-known interaction paradigms

![Diagram showing interaction paradigms and protocols]
Some informal semantics of interaction paradigms

Space coupling

CS

client

message(destination)

1.request-msg(destination)

server

2.reply-msg(source)

Time coupling

PS

publishers

2.event(topic)

broker

1.subscription(topic)

3.event(topic)

subscribers

Concurrency

TS

writers

1.tuple

tspace

2.tuple(template)

readers/takers
State-of-the-art in interaction paradigm interoperability

Theoretical approaches for individual paradigms
- Rely on concurrency theory, process algebras, architectural connectors
- No study of semantics across interconnected paradigms

Practical cross-paradigm approaches
- Typically apply to specific middleware implementations
- No assessment of end-to-end semantics
Solution to middleware interoperability

Generic Application (GA) connector

CS  PS  TS

JavaSpaces  Web Services

Planning & deployment

Higher-level connector

Interaction paradigm connectors

Middleware platforms

Info diffusion & coordination

PS connector

TS connector

Sensing

connector converter

GA connector
## Mapping of space coupling – IDLs

<table>
<thead>
<tr>
<th></th>
<th>CS</th>
<th>PS</th>
<th>TS</th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>element</strong></td>
<td>message</td>
<td>event</td>
<td>tuple</td>
<td>data</td>
</tr>
<tr>
<td><strong>main scope</strong></td>
<td>system ID</td>
<td>system ID</td>
<td>system ID</td>
<td>system ID</td>
</tr>
<tr>
<td><strong>sub-scope</strong></td>
<td>operation</td>
<td>topic</td>
<td>template</td>
<td>data qualifier</td>
</tr>
<tr>
<td><strong>interaction semantics</strong></td>
<td>{one-way, notification, request-response, solicit-response}</td>
<td>{publish, subscribe}</td>
<td>{write, take, read}</td>
<td>{post, get, post-get, get-post}</td>
</tr>
<tr>
<td><strong>S&amp;R element</strong></td>
<td>in: {sensorLocation, lifeSign}</td>
<td>out: {personnelId, personnelLocation}</td>
<td>out: {sensorLocation, lifeSign}</td>
<td>in/out data</td>
</tr>
</tbody>
</table>
Time coupling and concurrency of CS, PS, TS connectors

- Specify connector protocols with Labeled Transition Systems

- Express and verify semantics in LTL temporal logic

fluent SUBSCRIBED = \(<\{\text{subscribe}\}, \{\text{unsubscribe}\}>\)

assert EVENT_RECEIVED_IF_SUBSCRIBED_BEFORE_PUBLISH =
forall [e:EVENT] [] ((SUBSCRIBED && publish[e]) -> (!event_lost[e] U get_next_return[e]))
Mapping of time coupling and concurrency (I)

- Rely on the method of *protocol conversion via projections*\(^1\)
- Common semantics of CS, PS, TS apply to GA *end-to-end*

---

Protocol conversion via projections (I)

protocol P

Well-formed image protocol: all transitions are well-formed

image protocol P'

Safety and liveness properties of P' apply, properly inversely projected, also to P.
Protocol conversion via projections (II)

If I a meaningful protocol, then converter C_{A_1B_2} can be easily constructed: it statelessly maps between messages of A and B.

Properties of I apply to both A and B, and hence end-to-end (conversion correctness).

If no meaningful I, then a stateful converter C_{A_1B_2} may be intuitively constructed.

Properties of A apply to A_1A_C, but not necessarily inside A_C; symmetrically for B; (conversion correctness).
Our extension of the method

- Construction of a stateful converter not straightforward
- Projection and well-formedness check tedious for protocols with many states
- We guide the method based on knowledge of end-to-end protocols A and B

Looking for the converter $C_{A1B2}$

A good guess for the ‘interfaces’ of the converter towards $A_1$ and $B_2$

*Transformation of the problem* into finding $C_{AB}$.
We claim that $C_{AB}$ can be stateless and is straightforward to build.

Then, $C_{A1B2}$ can be easily derived.
Application of the method

CS connector

TS connector

common GA image

C_{CS-TS}

CS.receive_return.val1

TS.out.val1

r1.take_return.val1
Mapping of time coupling and concurrency (II)

- Certain semantics of CS, PS, TS not directly compatible
- Applications should appropriately constrain/complement the connector semantics
- Consider the combined semantics
Two coordinated TS readers/writers

Successful data delivery with coordination:

\[
\text{assert } \text{SENT\_DATA\_RECEIVED\_BY\_ALL\_ALWAYS} \\
= \forall d : \text{DATA} \ [ \ (\exists r : \text{READERS} \ [ \ !([\text{READERS}] \text{.take}[d] \cup ([r] \text{.read}[d] \parallel [r] \text{.take}[d]))) ] \]
\]

Successful data delivery without coordination:

\[
\text{fluent } \text{PENDING\_TAKE}[r : \text{READERS}] = < \{ [r] \text{.take} \}, \{ [r] \text{.take\_return}[d : \text{DATA}] \} > \\
\text{assert } \text{SENT\_DATA\_RECEIVED\_BY\_ALL\_IF\_ALL\_READ\_BEFORE\_TAKE} \\
= \forall d : \text{DATA} \ [ \ (\forall r : \text{READERS} \ [ \ !([\text{READERS}] \text{.take}[d] \cup ([r] \text{.read}[d] \parallel [r] \text{.take}[d]))) ] \implies \forall r : \text{READERS} \ [ \ !([\text{READERS}] \text{.take}[d] \cup ([r] \text{.read}[d] \parallel [r] \text{.take}[d]))) 
\]
Verification of GA end-to-end semantics

Successful data delivery for CS-to-PS interaction

\[
\text{assert } \overline{\text{SENT\_DATA\_RECEIVED\_IF\_SERVER\_ONLINE\_AND\_SUBSCRIBED\_BEFORE\_PUBLISH}} = \forall d : \text{DATA} \left[ \left( (\text{SERVER\_ONLINE} \land \text{SUBSCRIBED} \land \text{send}[d]) \rightarrow \left( (!\text{send\_fail}[d] \U \text{receive\_return}[d]) \land (!\text{event\_lost}[d] \U \text{get\_next\_return}[d])) \right) \right] 
\]

Successful data delivery for PS-to-TS interaction – TS constrained

\[
\text{assert } \overline{\text{SENT\_DATA\_RECEIVED\_BY\_ALL\_IF\_SUBSCRIBED\_BEFORE\_PUBLISH}} = \forall d : \text{DATA} \left[ \left( (\text{SUBSCRIBED} \land \text{publish}[d]) \rightarrow (\text{!event\_lost}[d] \U \text{get\_next\_return}[d]) \right) \land \left( \text{get\_next\_return}[d] \rightarrow \text{out}[d] \right) \land \left( \text{out}[d] \rightarrow \forall r : \text{READERS} \left( (!\text{READERS}.\text{take\_return}[d] \U ([r].\text{read\_return}[d] \lor [r].\text{take\_return}[d]))) \right) \right] 
\]
Implementation of the GA connector: an eXtensible Service Bus (XSB)

- XSB is a generic bus
- Built on top of a substrate bus
- Cross-paradigm integration in the S&R application

1 https://research.linagora.com/display/easyesb
Conclusion and outlook

- Service choreographies relying on heterogeneous middleware protocols
  - Require cross-middleware interoperability
- Tackle this challenge via
  - Formal abstractions and conversions for interaction paradigms
  - Assessment of end-to-end semantics
  - Implementation into an extensible service bus

Next steps
- Extend with support for continuous interactions – data streaming protocols
- Study non-functional properties across interaction paradigms
Thank you

Further information:

**XSB**: xsb.inria.fr

**CHORéOS**: www.choreos.eu

**Inria MiMove/ARLES**: https://mimove.inria.fr