Satisfying requirements for pervasive service compositions

Luca Cavallaro, Pete Sawyer, Daniel Sykes, Nelly Bencomo, Valérie Issarny

Lero, Lancaster, INRIA

Models @ Runtime, 2nd October 2012
Pervasive environment

• Cannot predict at design time what services will be available
• Services and devices appearing and disappearing all the time
• Huge variety of platforms, protocols, standards and functionality
• How do we **compose** services at **runtime** to achieve our aims?
TRAVEL BY CABRIOLET

Opera

Convert Sterling to eurodollars!

Orsay

Travel by Hansom cab

Louvre

Reserve a journey by balloon
Travel by cabriolet

Opera

Convert Sterling to eurodollars!

Orsay

Travel by Hansom cab

Louvre

TRAVEL BY CABRIOLET

Reserve a journey by balloon
Ticket booking app
(on smartphone)

chooseDate

chooseSeat

makeBooking

Ticket booking service
(on server)

pickSeat

selectDateTime

makeBooking
Ticket booking app  
(on smartphone)

Ticket booking service  
(on server)

chooseDate  

chooseSeat  

makeBooking  

pickSeat  

selectDateTime  

makeBooking  

Incompatible: signature & protocol mismatch
Possible solutions

• Standardisation of interfaces
  – So many standards
  – Often little incentive to standardise

• Manual translation between interfaces
  – Costly, slow
  – Cannot be applied at runtime
Runtime pervasive composition

• **Discover** services at runtime (WS-Discovery, UPNP)

• **Select** services relevant to **goal**

• Analyse their descriptions and **synthesize** a mediator

• Compose mediator with services to satisfy goal

• Adapt by replacing services that disappear
Runtime pervasive composition

Service 1  Goal  Service 2

Abstract mediator

Mediator synthesis

Runtime models

Running system

Service 1  Concrete mediator  Service 2

10
Goal model

executed(makeBooking)
## Service model

<table>
<thead>
<tr>
<th>Service 1</th>
<th>Category (e.g. “entertainment”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affordance (ontology concept)</td>
<td>Available operations (e.g. “getTicketPrice”) labelled with ontology concepts</td>
</tr>
<tr>
<td>Interface (WSDL)</td>
<td>Protocol describing valid combinations of operations</td>
</tr>
<tr>
<td>Behaviour (state machine)</td>
<td></td>
</tr>
</tbody>
</table>
Service selection

Service 1
- Affordance (ontology concept)
- Interface (WSDL)
- Behaviour (state machine)

Service 2
- Affordance (ontology concept)
- Interface (WSDL)
- Behaviour (state machine)

Mediator synthesis

Ontology matching
Mediator synthesis

• Synthesise intermediary that enables communication between two differing protocols

• Such that the goal formula is achieved
  \[ P_{S_1} \times M \times P_{S_2} \models G \]

• Simple goal language: LTL operators plus
  – sent(c), received(c), executed(c)
  – Parameter and operation concepts in ontology
  – \(<>executed(makeBooking)\)
Mediator synthesis

• Goal LTL compiled to Büchi automaton, reachability checked on parallel composition

• Path to goal must be a feasible interaction:
  – All input parameters are sent from one partner before being needed by the other partner
  – All output parameters are eventually provided
  – Permits operation re-ordering
• Services effectively synchronise on operations with matching ontology concepts

Ontology:
- DateTimeOperation
  - chooseDate
  - selectDateTime

Feasible interaction in mediator
- DateTime chooseDate()
- void selectDateTime(DateTime t)
Ticket booking app (on smartphone)

chooseDate

chooseSeat

makeBooking

Mediator

>chooseDate

>chooseSeat

pickSeat>

selectDateTime>

makeBooking>

Ticket booking service (on server)

pickSeat

selectDateTime

makeBooking

<>executed(makeBooking)
Ticket booking app
(on smartphone)

Ticket booking service
(on server)

chooseDate

chooseSeat

makeBooking

>chooseDate

>chooseSeat

pickSeat>

selectDateTime>

makeBooking>

<>executed(makeBooking)
Ticket booking app
(on smartphone)

chooseDate

chooseSeat

makeBooking

Mediator

>chooseDate

>chooseSeat

pickSeat>

selectDateTime>

makeBooking>

Ticket booking service
(on server)

pickSeat

selectDateTime

makeBooking

<>executed(makeBooking)
Ticket booking app
(on smartphone)

chooseDate

chooseSeat

makeBooking

->chooseDate

->chooseSeat

->makeBooking

<>executed(makeBooking)

Mediator

>chooseDate

>chooseSeat

pickSeat>

selectDateTime>

makeBooking>

Ticket booking service
(on server)

pickSeat

selectDateTime

makeBooking
Summary

• Build compositions of multiple services discovered at runtime
• Services describe their interface and behaviour (runtime models)
• Synthesis overcomes signature and protocol mismatch
• Achieve goals specified using KAOS
• Future:
  – Consider non-functional properties
  – Relaxed goals guided by what can be realised given discovered services