Supporting Test-Driven Development of Web Services Choreographies

Pedro Leal, Felipe Besson, Fabio Kon, Alfredo Goldman
University of São Paulo

Dejan Milojicic
HP Labs

This research is funded by:

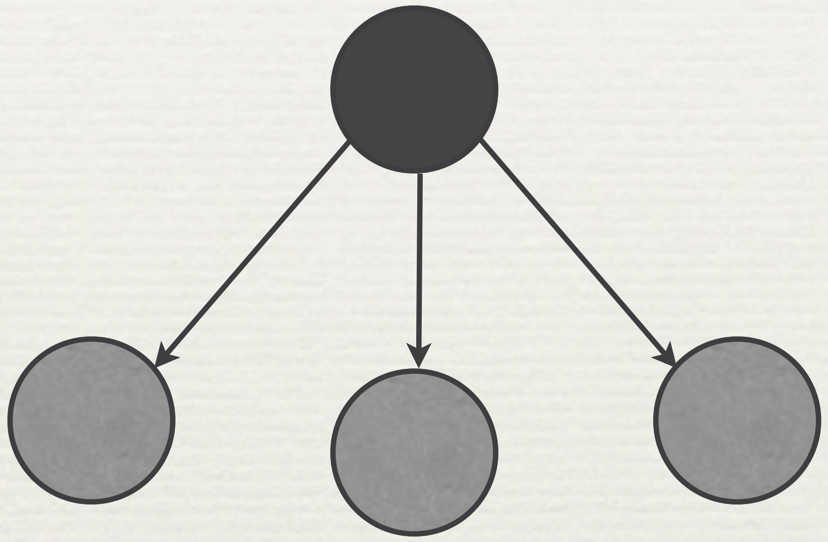
EUROPEAN COMMISSION

CHOReOS
Large Scale Choreographies for the Future Internet

Web Service Compositions

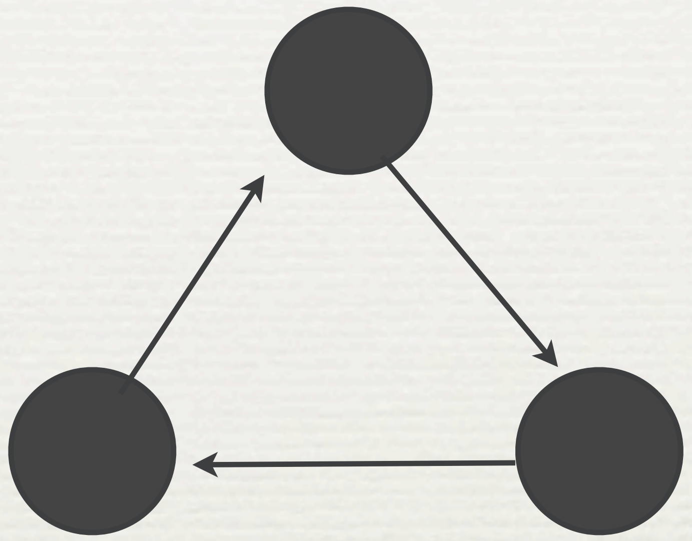
- One of the Service-Oriented Architecture Principles
- Deliver a service with the collaboration of a set of web services
- Two main approaches:
  - Orchestration
  - Choreography

Orchestration



Process of coordinating an exchange of information through web service interactions

Choreography



· Describes the flow of messages between a set of services in a global choreography, without a controller

· Services act as peers, interactions are long-lived & stateful

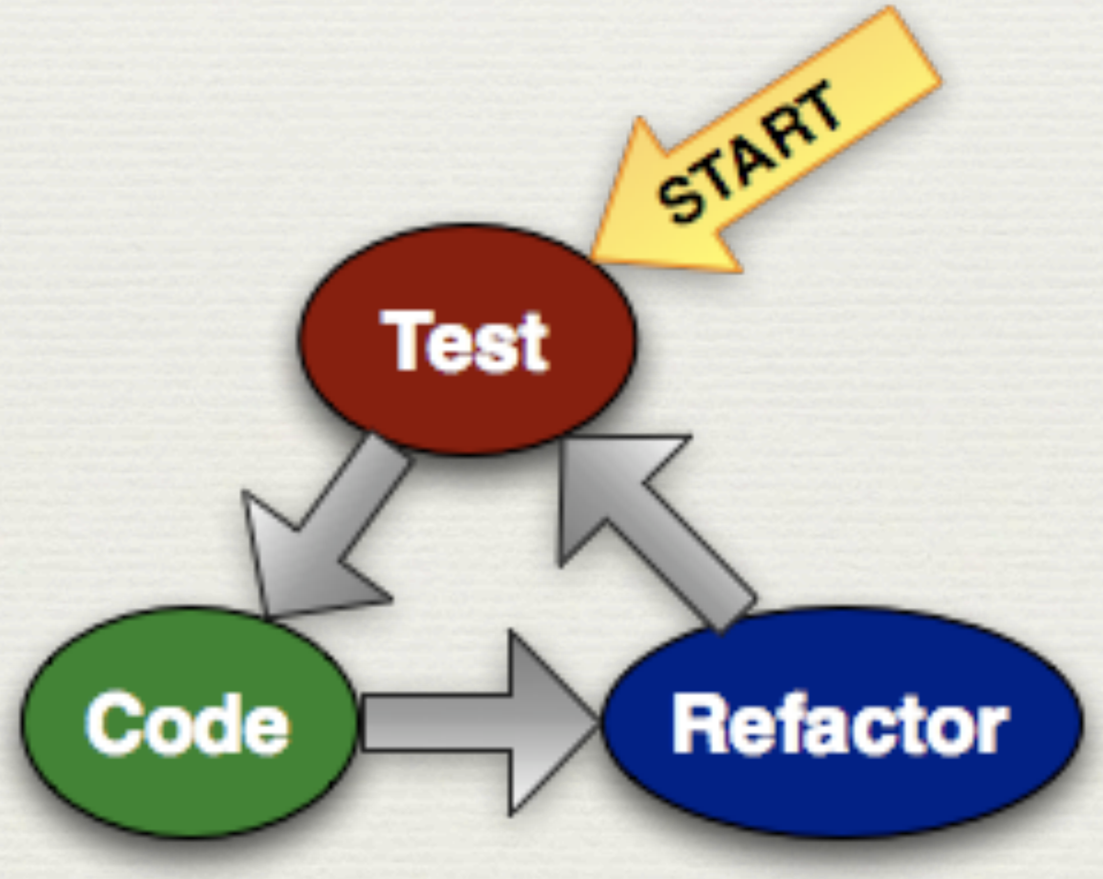
Goal

Test-Driven Development (TDD) of web service choreographies

Tools that deploy and enact ws choreographies on a cloud environment, e.g., Open Cirrus or Amazon EC2

Test-Driven Development

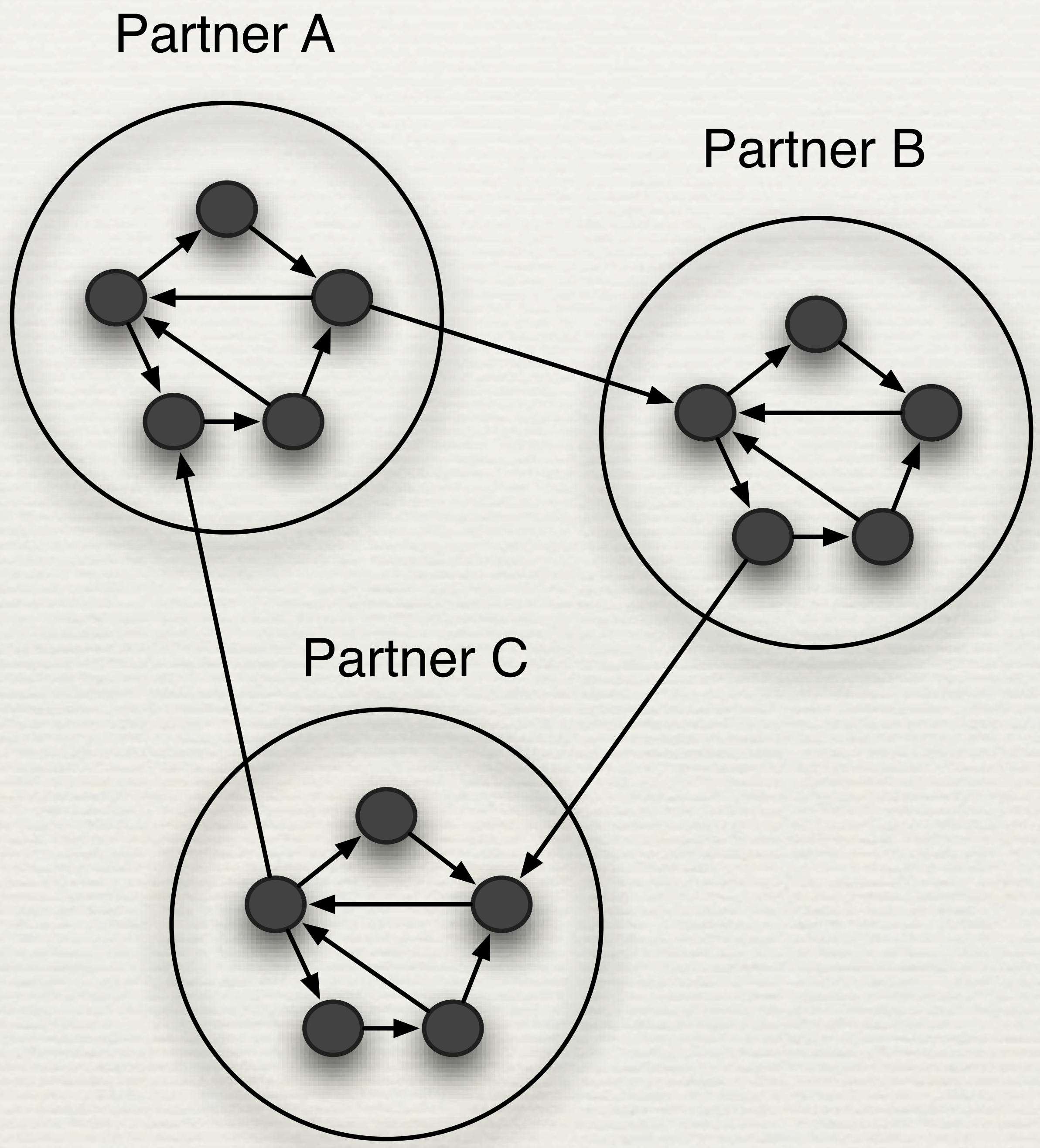
A design technique that drives the development process through testing (Fowler, 2011; Beck, 2002)



Web Service Choreography Testing Challenges (Canfora, 2009)

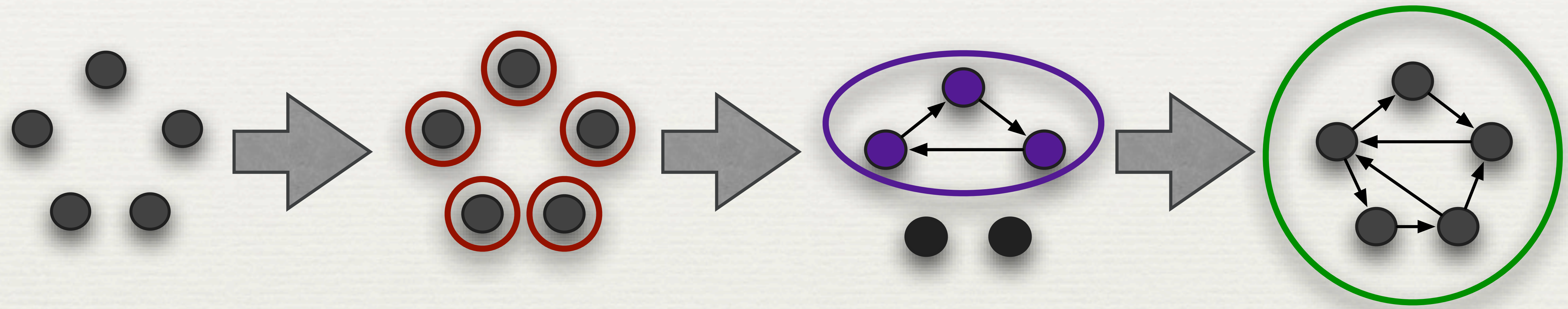
- Dinamicity and adaptiveness
- Decentralized flow of information
- No widely-adopted standards
- Third-party service issues

Testing Techniques



Web Service Choreography

Testing Techniques



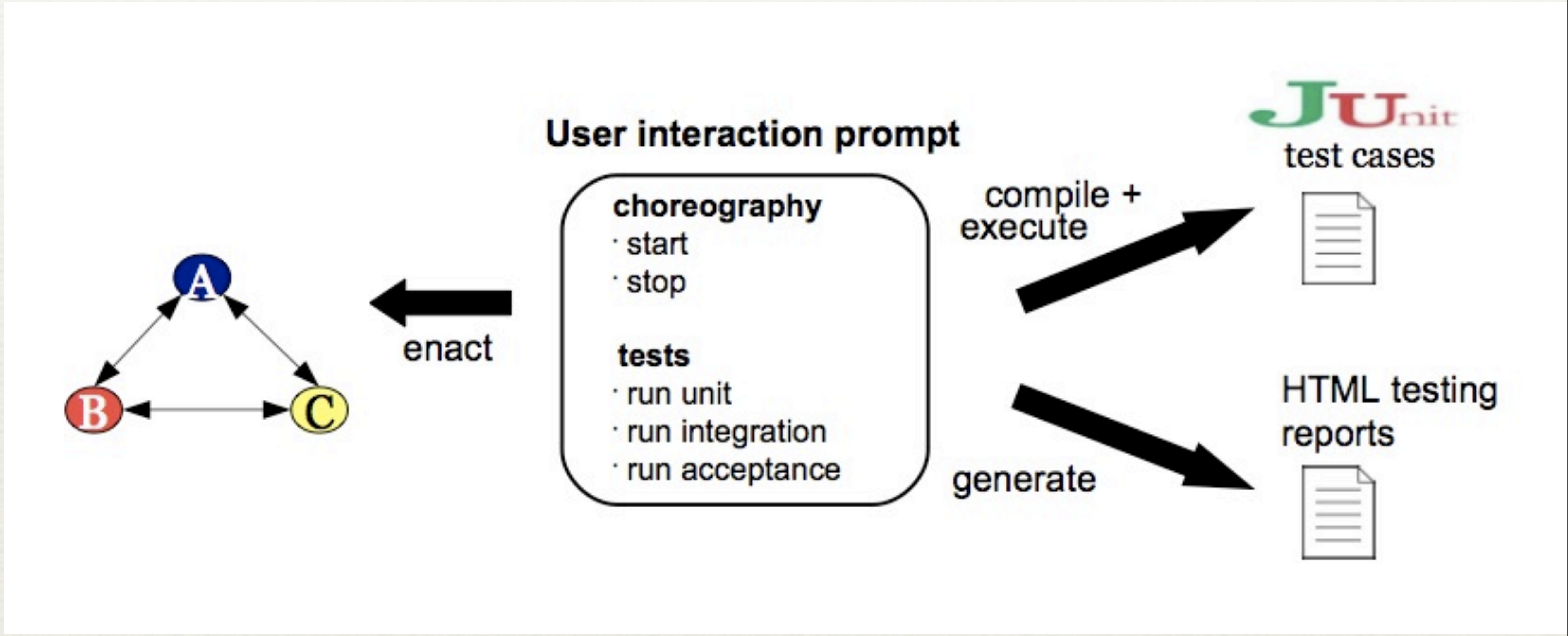
● Web Services

○ Unit Tests

○ Integration Tests

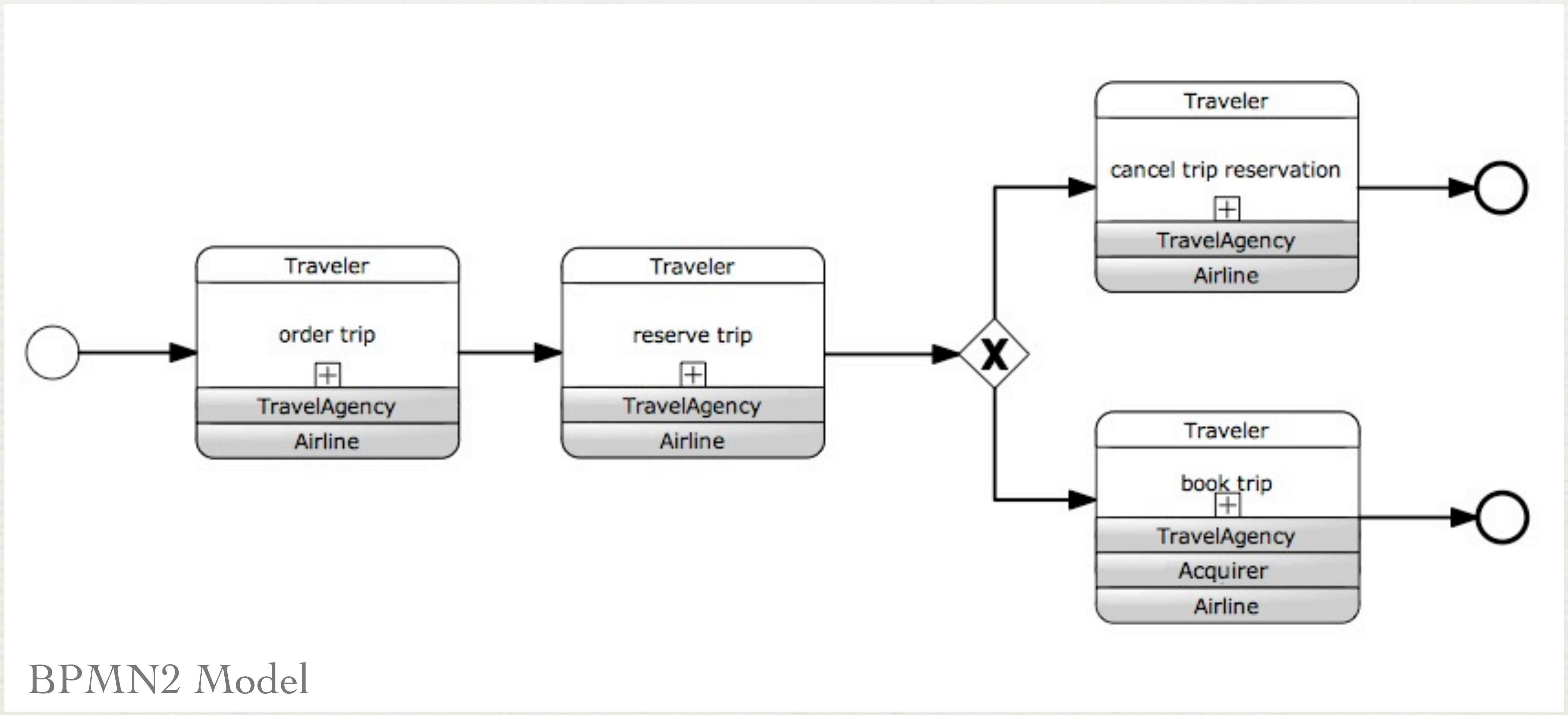
○ Acceptance Tests

Prototype



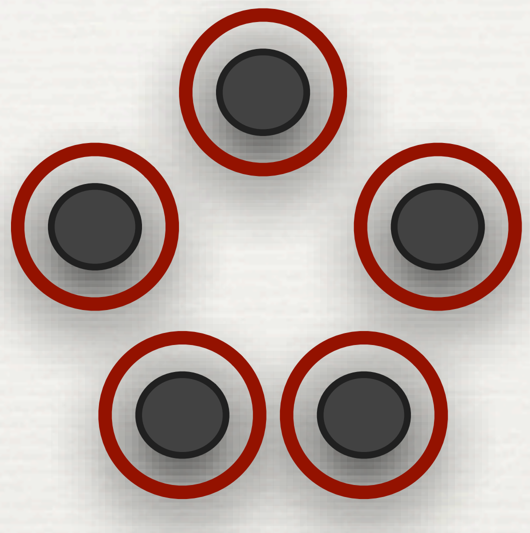
- Ad hoc bash scripts for choreography deployment and enactment
- JUnit framework to support the automated testing

Web Service Choreography Example



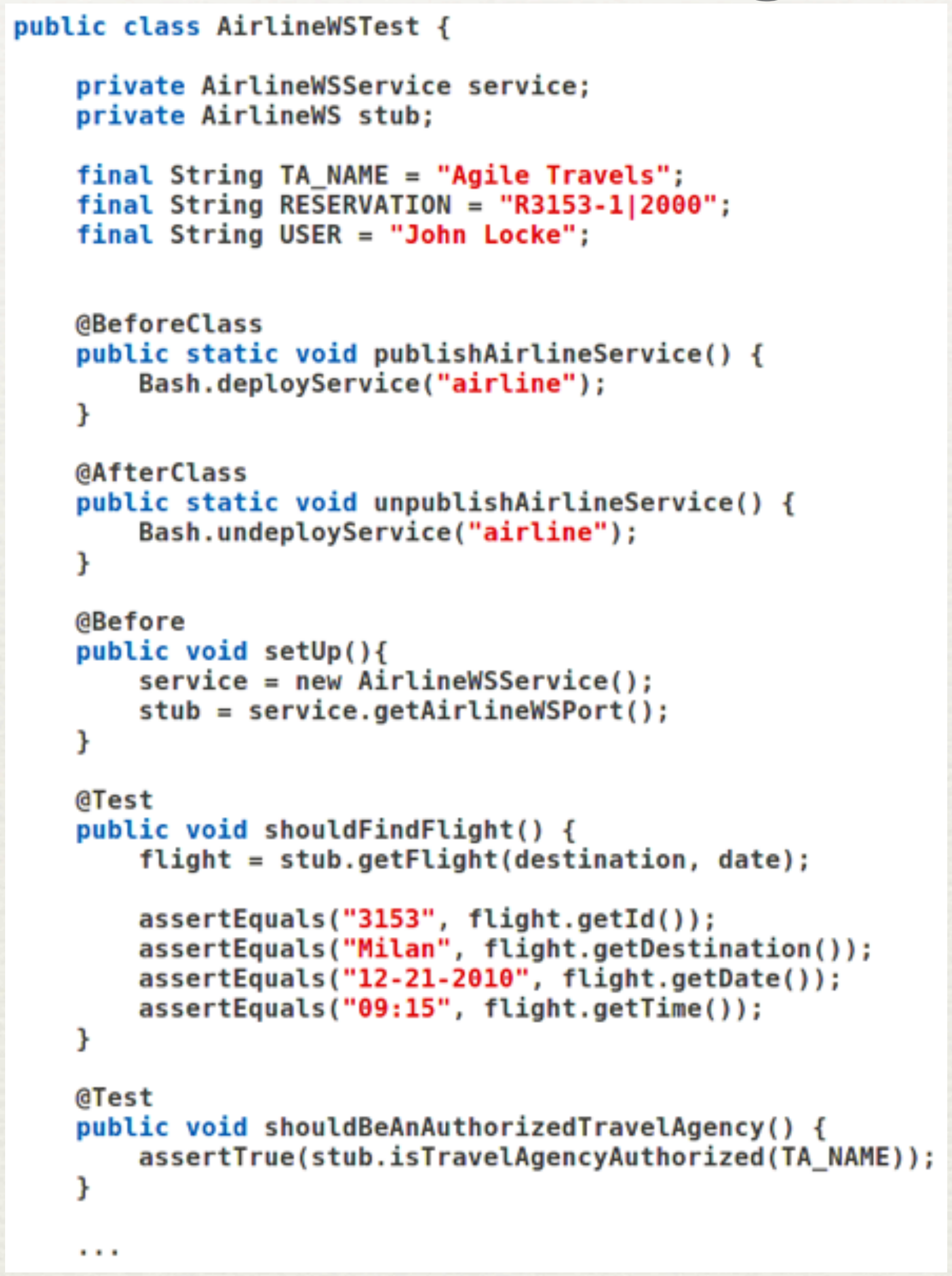
With OpenKnowlegde (OK, 2011)

Unit Testing

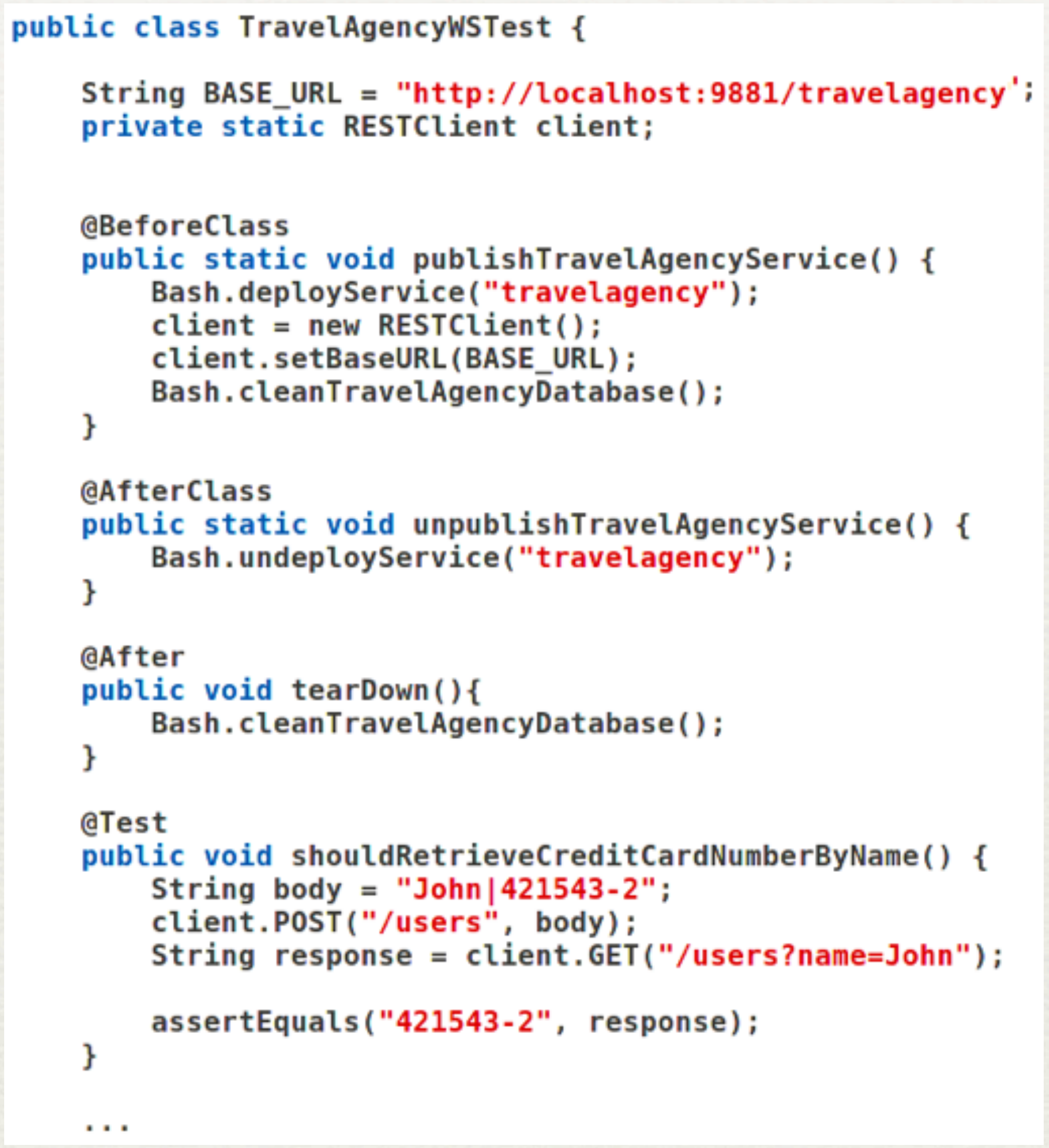


- Test each web service of the choreography
- Script to deploy the web service
- Client to communicates with each one

Unit Testing



```
public class AirlineWSTest {

    private AirlineWSService service;
    private AirlineWS stub;

    final String TA_NAME = "Agile Travels";
    final String RESERVATION = "R3153-1|2000";
    final String USER = "John Locke";


    @BeforeClass
    public static void publishAirlineService() {
        Bash.deployService("airline");
    }

    @AfterClass
    public static void unpublishAirlineService() {
        Bash.undeployService("airline");
    }

    @Before
    public void setUp(){
        service = new AirlineWSService();
        stub = service.getAirlineWSPort();
    }

    @Test
    public void shouldFindFlight() {
        flight = stub.getFlight(destination, date);

        assertEquals("3153", flight.getId());
        assertEquals("Milan", flight.getDestination());
        assertEquals("12-21-2010", flight.getDate());
        assertEquals("09:15", flight.getTime());
    }

    @Test
    public void shouldBeAnAuthorizedTravelAgency() {
        assertTrue(stub.isTravelAgencyAuthorized(TA_NAME));
    }

    ...
```

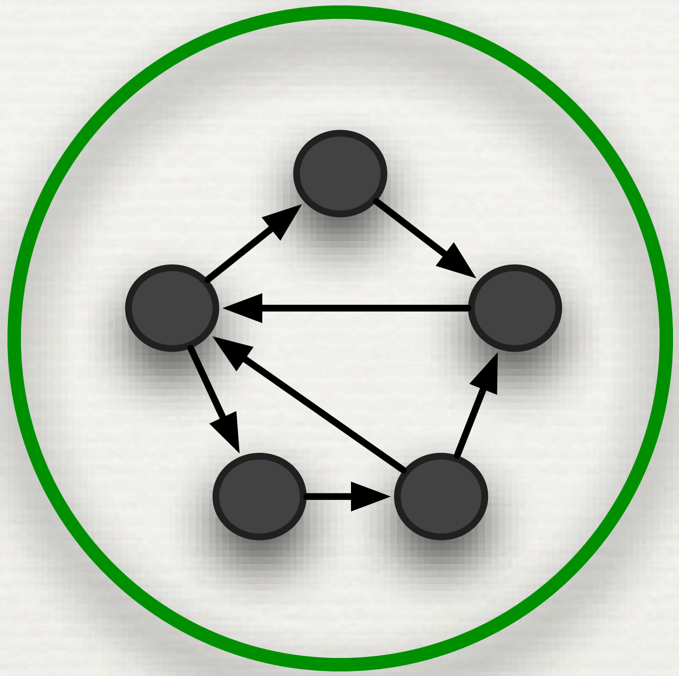


```
public class TravelAgencyWSTest {

    String BASE_URL = "http://localhost:9881/travelagency";
    private static RESTClient client;


    @BeforeClass
    public static void publishTravelAgencyService() {
        Bash.deployService("travelagency");
        client = new RESTClient();
        client.setBaseURL(BASE_URL);
        Bash.cleanTravelAgencyDatabase();
    }

    @AfterClass
    public static void unpublishTravelAgencyService() {
        Bash.undeployService("travelagency");
    }

    @After
    public void tearDown(){
        Bash.cleanTravelAgencyDatabase();
    }

    @Test
    public void shouldRetrieveCreditCardNumberByName() {
        String body = "John|421543-2";
        client.POST("/users", body);
        String response = client.GET("/users?name=John");

        assertEquals("421543-2", response);
    }

    ...
```

SOAP WS

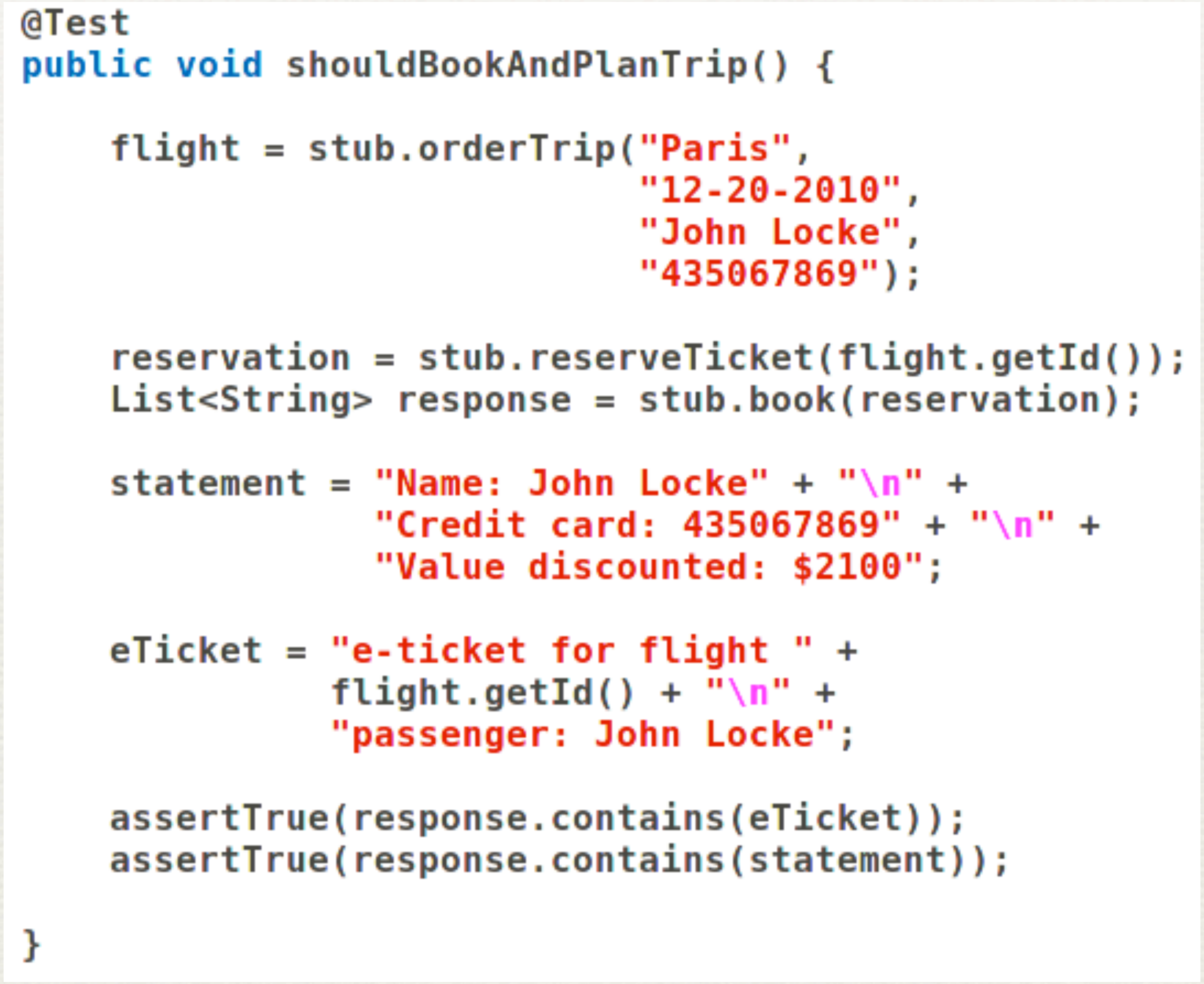
REST WS

Acceptance Testing



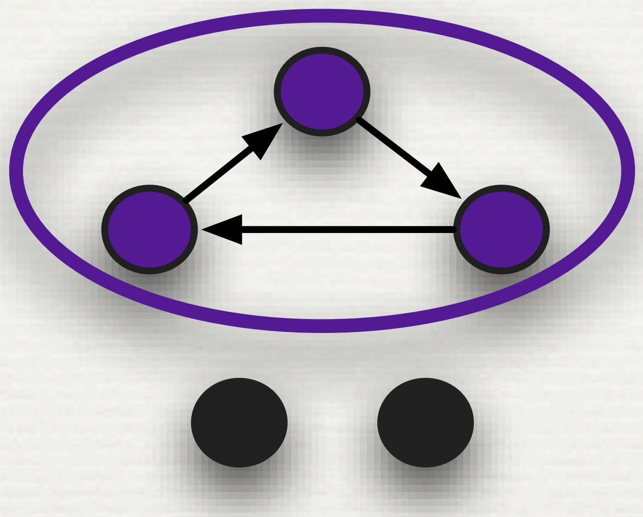
- Similar to Unit Testing
- Test the web service choreography as an atomic service
- Script to deploy and enact the web service choreography

Acceptance Testing



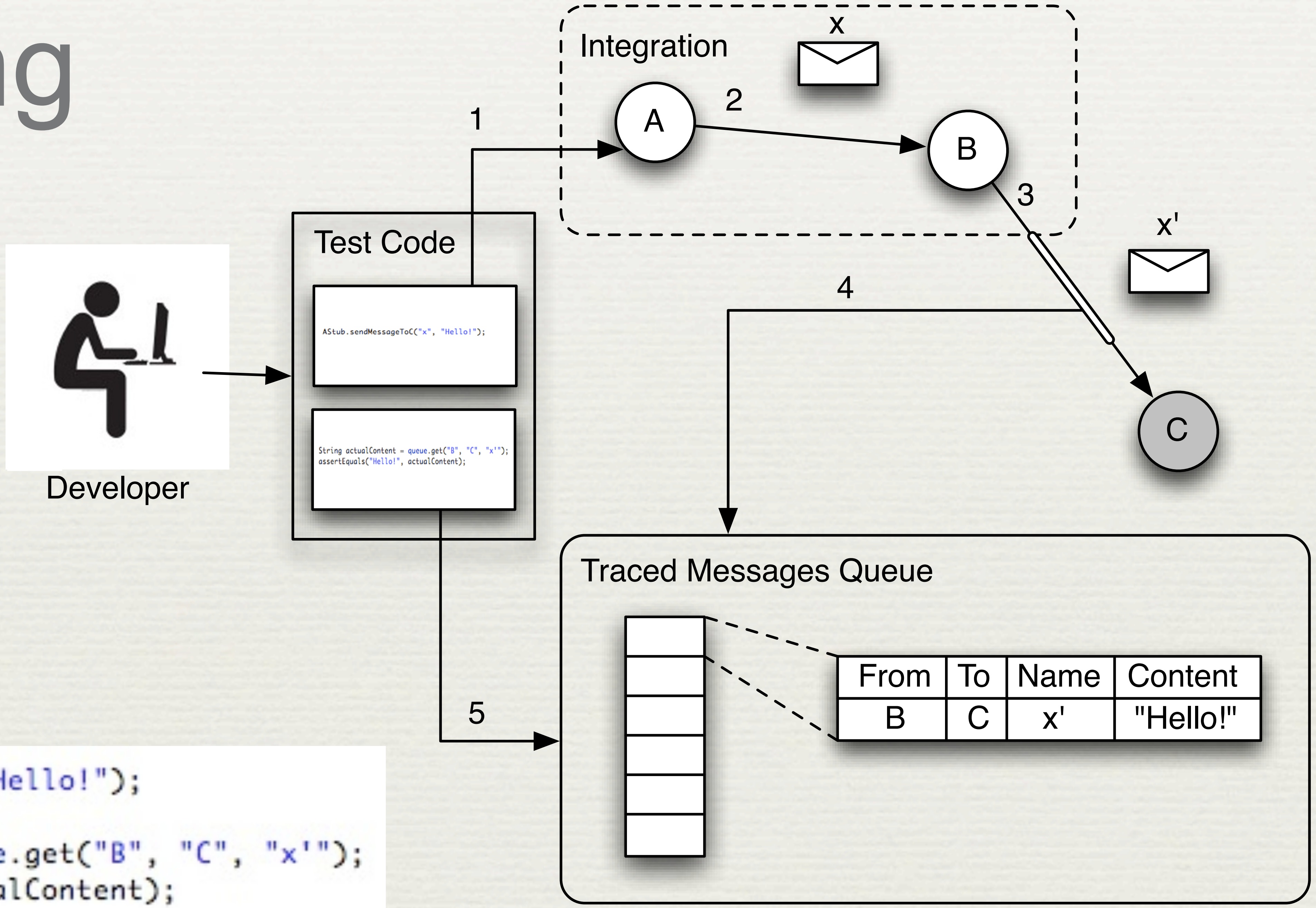
```
@Test
public void shouldBookAndPlanTrip() {

    flight = stub.orderTrip("Paris",
                            "12-20-2010",
                            "John Locke",
                            "435067869");

    reservation = stub.reserveTicket(flight.getId());
    List<String> response = stub.book(reservation);

    statement = "Name: John Locke" + "\n" +
                "Credit card: 435067869" + "\n" +
                "Value discounted: $2100";

    eTicket = "e-ticket for flight " +
              flight.getId() + "\n" +
              "passenger: John Locke";

    assertTrue(response.contains(eTicket));
    assertTrue(response.contains(statement));

}
```

Integration Testing

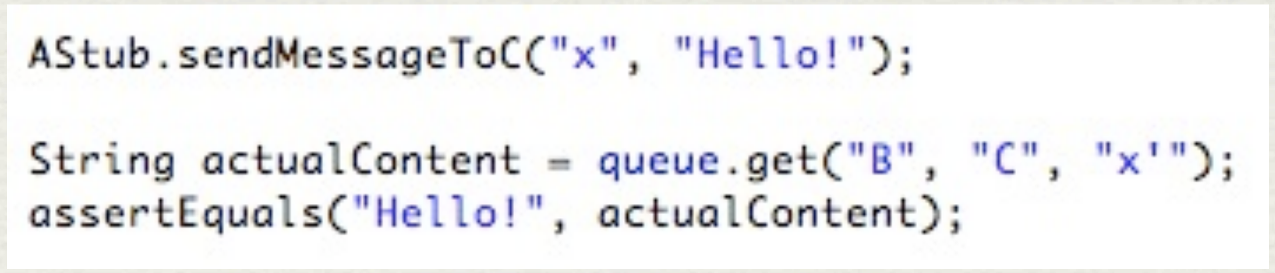


- Need to verify the interaction among the web services
- We will evaluate the messages exchanged
- Verify their output messages when they integrate with the choreography

Integration Testing


Integration
x
1
2
A
B
3
x'
4
C
Test Code
AStub.sendMessageToC("x", "Hello!");
String actualContent = queue.get("B", "C", "x'");
assertEquals("Hello!", actualContent);
Developer
5
Traced Messages Queue

From	To	Name	Content
B	C	x'	"Hello!"



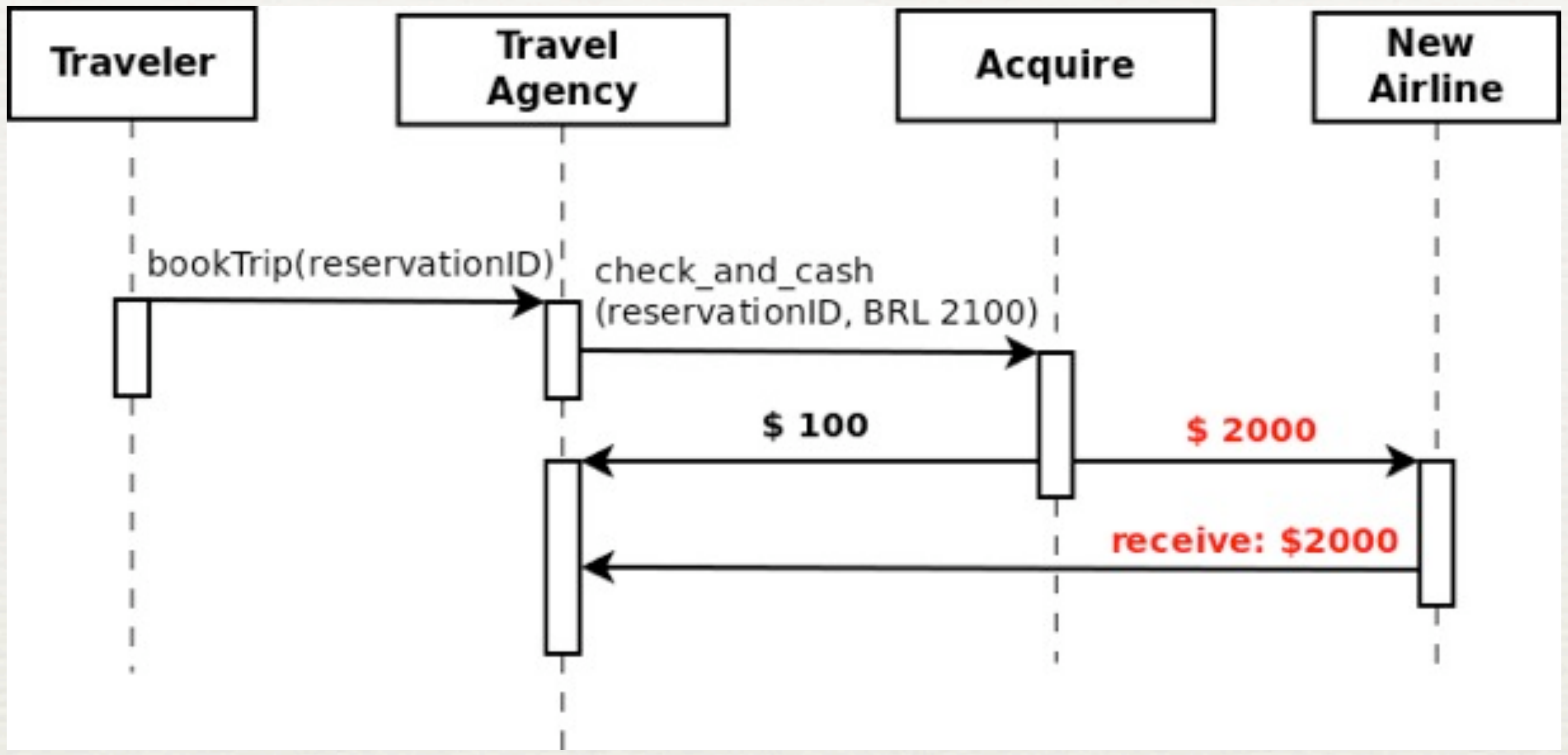
```
AStub.sendMessageToC("x", "Hello!");

String actualContent = queue.get("B", "C", "x'");
assertEquals("Hello!", actualContent);
```

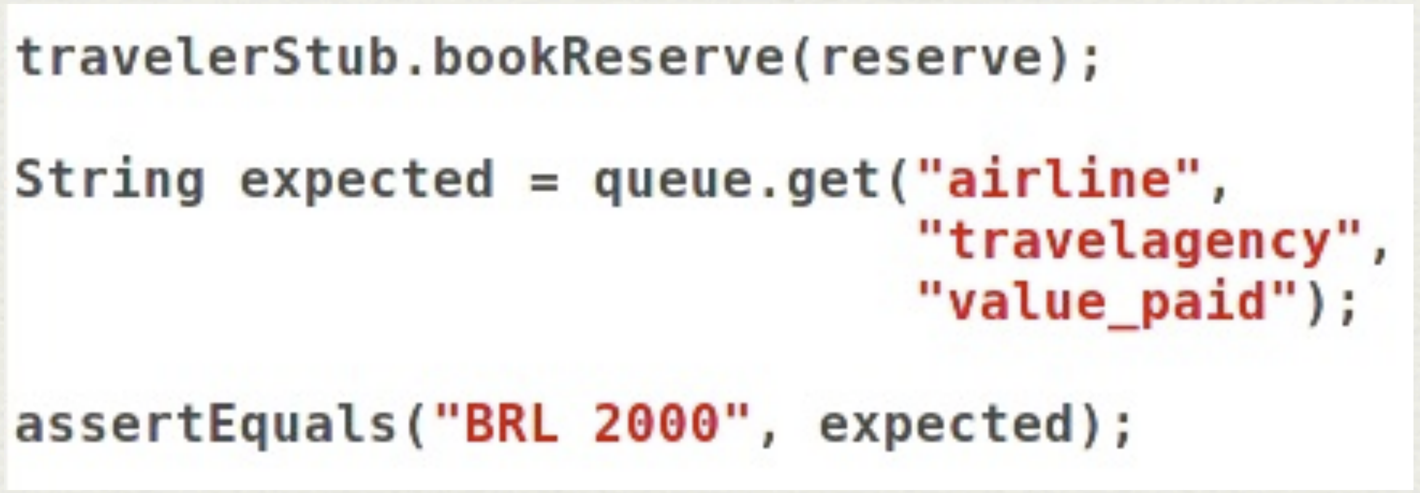
Existing Tools

Web service testing tool: SoapUI

- Generates SOAP clients automatically
- Must fill XML-Soap envelope
- Does not support integration tests

Integration Testing Example





```
travelerStub.bookReserve(reserve);

String expected = queue.get("airline",
                            "travelagency",
                            "value_paid");

assertEquals("BRL 2000", expected);
```

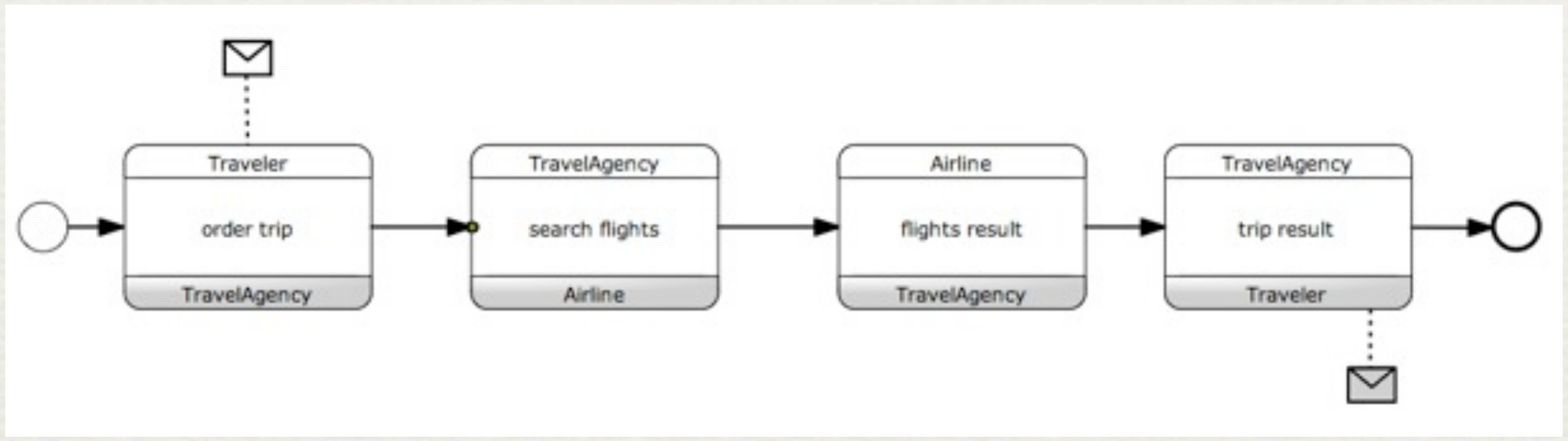
Example of a bug discovered by the integration testing technique

Evaluating the Overheads

- The procedure of collecting messages exchanged might cause an overhead

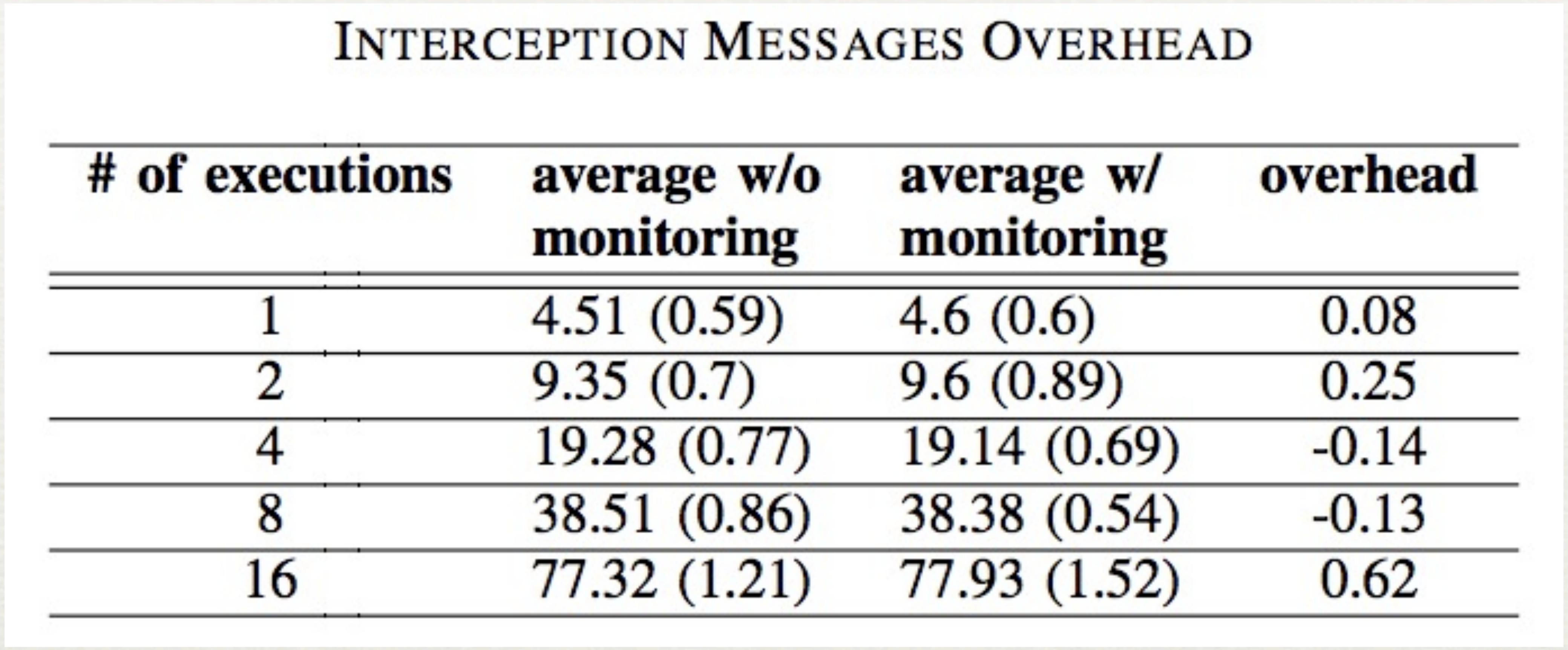
Evaluating the Overheads

- Compare the execution time of a choreography functionality using and not using our approach
- Simple interaction operation - 4 messages exchanged



- Each web service was allocated on a dedicated note of a cluster
- Execution time for 1, 2, 4, 8, and 16 sequential executions
- For each one, we collected 30 samples

Evaluating the Overheads


INTERCEPTION MESSAGES OVERHEAD

# of executions	average w/o monitoring	average w/ monitoring	overhead
1	4.51 (0.59)	4.6 (0.6)	0.08
2	9.35 (0.7)	9.6 (0.89)	0.25
4	19.28 (0.77)	19.14 (0.69)	-0.14
8	38.51 (0.86)	38.38 (0.54)	-0.13
16	77.32 (1.21)	77.93 (1.52)	0.62

- Overhead smaller than the standard deviations
- Overhead negligible
- The stored messages are simple
- We intend to evaluate the overhead behavior in a more realistic choreography on cloud environments

Ongoing Work

Improvements for the testing framework:

- Generating web service clients dynamically
- Manipulating the elements of a choreography more easily
- Mocking third-party services
- Improving the interception of exchanged messages
- Automating the deployment and enactment of a choreography on a cloud environment

Questions?

This research is funded by:

EUROPEAN COMMISSION

CHOReOS Large Scale Choreographies for the Future Internet

More information on: http://ccsl.ime.usp.br/baile/VandV

Pedro Leal
pedrombl@ime.usp.br

References

- Martin Fowler. Test-Driven Development. 2011. Available on: http://www.martinfowler.com/bliki/TestDrivenDevelopment.html
- Kent Beck. Test Driven Development: By Example. Addison-Wesley Professional, 2002.
- Gerardo Canfora and Massimiliano Di Penta. Service-oriented architectures testing: A survey. In Andrea De Lucia and Filomena Ferrucci, editors, Software Engineering, volume 5413 of Lecture Notes in Computer Science, pages 78–105. Springer Berlin / Heidelberg, 2009.
- OpenKnowledge (OK). 2011. Available on: <http://www.openk.org>
- SoapUI (SoapUI). 2011. Available on: <http://www.soapui.org/>