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:
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

BPMN2 Model

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
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 shouldRetrieveCreditCardNumberByMail() {
        String body = "John|421543-2";
        client.POST("/users", body);
        String response = client.GET("/users?name=John");

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

    @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");
    }

    @Before
    public void setUp(){
        Bash.undeployService("travelagency");
    }

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

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

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

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

    ...
}
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

```java
@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

Test Code

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
Example of a bug discovered by the integration testing technique
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

- 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:

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

Pedro Leal
pedrombl@ime.usp.br
References