System Integration Report

Online Learning Platform

AUTHOR: ÅSA WEGELIUS, CLOVIS LEBRET, TUDOR STOICA

OWNER: ÅSA WEGELIUS

CLIENT: JARL TUXEN

VERSION: 1.0.5

1. SystemIntegrationReport Document History

1.2 Revision History

version	Revision	date	Implemented by
1.0	added part about ORM	13-05-16	Åsa Wegelius
1.1	added intro and a small part about jersey	28/05/2016	Clovis Lebret
1.2	added conclusion and added a small part in integration patterns	30/05/2016	Clovis Lebret
1.3	completed the report	30/05/2016	Clovis Lebret

1.3 Approvals

version	Name	title	Date

1.4 Distribution

version	Name	title	Date

1.5 Confidentiality Rating

Rating	
Company Confidential	x
Non Confidential	

2. Table of Contents

3. Introduction

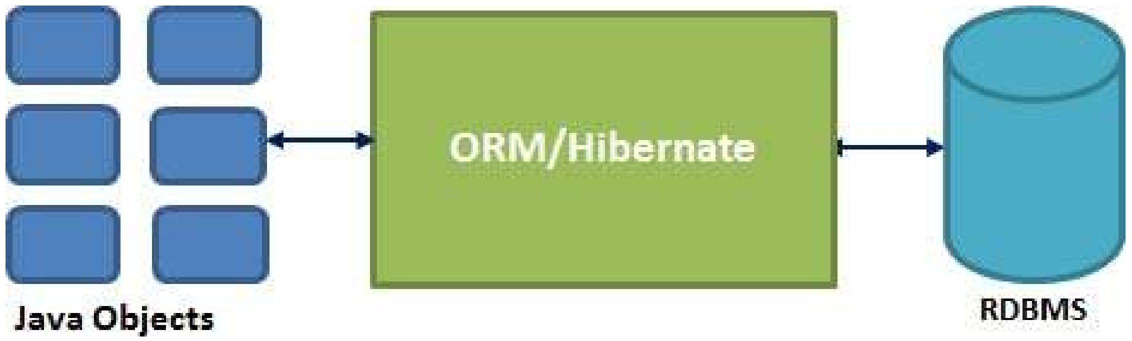
This report documents the development process and our learning experiences over the course of implementing our project for the course *System Integration* , as part of the Software Development education at The Copenhagen School of Design and Technology.

Through this report we explain how we implemented system integration techniques and technologies within the context of our project. In each section, we will provide a brief introduction to the theory behind the specific topic to give a better understanding of our reasoning and the choices we have made throughout the development, particularly in regards to our design and implementation as it relates to achieving particular integration goals, and what tradeoffs, if any, were made.

4. Integration

4.1 Object Relational Mapping

In object-oriented programming you work on Objects that are almost always non-scalar values. In a relational DBMS you store and manipulate scalar values in tables. To integrate you must either convert the objects to groups of simple values for storage and then convert back upon retrieval or you can use only simple scalar values within the program. Object-relational mapping implements the first approach.



Hibernate mapping

We use Hibernate ORM as framework. Hibernates core features are mapping from Java classes to database tables and from Java data types to SQL data types. Some of the advantages to use Hibernate: - It is database independent (no need for database specific queries and syntax). You use Hibernate Query Language (HQL). - You will get all advantages of OOP concepts like inheritance, encapsulation etc. - It provides caching mechanism (1st level & 2nd level cache) so you don't need to hit database for similar queries, you can cache it and use it from buffered memory to improve performance. - It supports Lazy loading. That is, if parent class have n + 1 children and you only need information about one child there is no need to load the n children. Load only what you need.

To get up and running with Hibernate you need a configuration file (hibernate.cfg.xml), a sessions factory and then you either use annotations for mapping or xml mapping files.

hibernate.cfg.xml:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/onlinelearningplatf
orm</property>
    <property name="hibernate.connection.username">*****</property>
    <property name="hibernate.connection.password">******</property>
    <property name="hibernate.current_session_context_class">thread</property>
    <mapping
resource="se/wegelius/olpstudenthandler/model/persistance/ContentProviderPersis
tance.hbm.xml"/>
    <mapping
resource="se/wegelius/olpstudenthandler/model/persistance/CourseBranchPersistan
ce.hbm.xml"/>
    <mapping
resource="se/wegelius/olpstudenthandler/model/persistance/CoursePersistance.hbm
.xml"/>
    <mapping
resource="se/wegelius/olpstudenthandler/model/persistance/CourseTypePersistance
.hbm.xml"/>
    <mapping
resource="se/wegelius/olpstudenthandler/model/persistance/LecturePersistance.hb
m.xml"/>
    <mapping
resource="se/wegelius/olpstudenthandler/model/persistance/MultipleChoiceAnswerP
ersistance.hbm.xml"/>
    <mapping
resource="se/wegelius/olpstudenthandler/model/persistance/MultipleChoiceOptions
Persistance.hbm.xml"/>
    <mapping
resource="se/wegelius/olpstudenthandler/model/persistance/MultipleChoiceQuestio
nPersistance.hbm.xml"/>
    <mapping
resource="se/wegelius/olpstudenthandler/model/persistance/VerificationtokenPers
istance.hbm.xml"/>
    <mapping
resource="se/wegelius/olpstudenthandler/model/persistance/UserPersistance.hbm.x
ml"/>
    <mapping
resource="se/wegelius/olpstudenthandler/model/persistance/PlaylistPersistance.h
bm.xml"/>
  </session-factory>
</hibernate-configuration>
```

sessions factory:

```
public class HibernateUtil {
    private static SessionFactory sessionFactory;

    public static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            // loads configuration and mappings
            Configuration configuration = new
Configuration().configure();
            ServiceRegistry serviceRegistry
                = new StandardServiceRegistryBuilder()

.applySettings(configuration.getProperties()).build();

            // builds a session factory from the service registry
            sessionFactory =
configuration.buildSessionFactory(serviceRegistry);
        }

        return sessionFactory;
    }
}
```

xml mapping file:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<!-- Generated May 10, 2016 4:15:18 PM by Hibernate Tools 4.3.1 -->
<hibernate-mapping>
    <class
name="se.wegelius.olpstudenthandler.model.persistance.CourseTypePersistance"
table="course_type" catalog="onlinelearningplatform">
        <id name="courseTypeId" type="java.lang.Integer">
            <column name="course_type_id" />
            <generator class="identity"></generator>
        </id>
        <property name="courseTypeName" type="string">
            <column name="course_type_name" length="45" unique="true" />
        </property>
        <property name="ctCourseBranchFk" type="java.lang.Integer">
            <column name="ct_course_branch_fk" />
        </property>
        <set name="courses" table="course" inverse="true" lazy="true"
fetch="select">
            <key>
                <column name="c_course_type_fk" not-null="true" />
            </key>
            <one-to-many
class="se.wegelius.olpstudenthandler.model.persistance.CoursePersistance" />
        </set>
    </class>
</hibernate-mapping>
```

persistance class:

```
public class CourseTypePersistance  implements java.io.Serializable {


     private Integer courseTypeId;
     private String courseTypeName;
     private Integer ctCourseBranchFk;
     private Set<CoursePersistance> courses = new HashSet<CoursePersistance>(0);

    public CourseTypePersistance() {
    }

    public CourseTypePersistance(String courseTypeName, Integer
ctCourseBranchFk, Set<CoursePersistance> courses) {
       this.courseTypeName = courseTypeName;
       this.ctCourseBranchFk = ctCourseBranchFk;
       this.courses = courses;
    }

    public Integer getCourseTypeId() {
        return this.courseTypeId;
    }

    public void setCourseTypeId(Integer courseTypeId) {
        this.courseTypeId = courseTypeId;
    }
    public String getCourseTypeName() {
        return this.courseTypeName;
    }

    public void setCourseTypeName(String courseTypeName) {
        this.courseTypeName = courseTypeName;
    }
    public Integer getCtCourseBranchFk() {
        return this.ctCourseBranchFk;
    }

    public void setCtCourseBranchFk(Integer ctCourseBranchFk) {
        this.ctCourseBranchFk = ctCourseBranchFk;
    }
    public Set<CoursePersistance> getCourses() {
        return this.courses;
    }

    public void setCourses(Set<CoursePersistance> courses) {
        this.courses = courses;
    }
}
```

One advantage we got from using Hibernate was that we could write a generic dao. This cuts down the amount of dao code we need to write considerable. Most of it is reusable code so you can reap the benefits in future projects.

the generic interface

```
public interface IOlpDao<T, ID extends Serializable> {
    Class<T> getEntityClass();
    T findByID(ID id);
    Set<T> getAll();
    Set<T> query(String hsql, Map<String, Object> params);
    int count();
    void save(T entity);
    void update(T entity);
    void saveOrUpdate(T entity);
    void merge(T entity);
    void delete(T entity);
}
```

example from the interface implementation:

```
    @Override
    public void save(T entity) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        try {
            session.beginTransaction();
            session.save(entity);
            session.getTransaction().commit();
        } catch (HibernateException e) {
            logger.error("tried to save " + e.getMessage());
            e.printStackTrace();
        } finally {
            try {
                session.close();
            } catch (HibernateException e) {
                logger.error(e.getMessage());
            }
        }
    }


    @SuppressWarnings("unchecked")
    @Override
    public Set<T> query(String hsql, Map<String, Object> params) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        List<T> objects = null;
        logger.info("hsql: " + hsql + " params size: " + params.size());
        try {
            session.beginTransaction();
            Query query = session.createQuery(hsql);
            if (params != null) {
                for (String i : params.keySet()) {
                    query.setParameter(i, params.get(i));
                }
            }
            logger.info("query parameters: " +
Arrays.toString(query.getNamedParameters()));
            if ((!hsql.toUpperCase().contains("DELETE"))
                    && (!hsql.toUpperCase().contains("UPDATE"))
                    && (!hsql.toUpperCase().contains("INSERT"))) {
                objects = query.list();
                logger.info("FINISHED - query. Result size=" + objects.size());
            } else {
                logger.info("FINISHED - query. ");
            }
            session.getTransaction().commit();
        } catch (HibernateException e) {
            logger.error(e.getMessage());

            e.printStackTrace();
        } finally {
            try {
                session.close();
            } catch (HibernateException e) {
                logger.error(e.getMessage());
            }
        }
        if (objects != null) {
            logger.info("no of objects: " + objects.size());
            return new HashSet<>(objects);
        }
        return null;
    }
```

example from CourseDao:

```
public class CourseDao extends OlpDao<CoursePersistance, Integer> {

    private static final org.slf4j.Logger logger =
LoggerFactory.getLogger(CourseDao.class);

    public CourseDao(Class<CoursePersistance> type) {
        super(type);
    }

    /**
     *
     */
    public CourseDao() {
        super(CoursePersistance.class);
    }

    /**
     * Find an entity by its primary key
     *
     * @param id the entity's primary key
     * @return the entity
     */
    @SuppressWarnings("unchecked")
    @Override
    public CoursePersistance findByID(Integer id) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        CoursePersistance course = null;
        try {
            session.beginTransaction();
            course = (CoursePersistance) session.get(CoursePersistance.class,
id);
            Hibernate.initialize(course.getCourseBranch());
            Hibernate.initialize(course.getContentProvider());
            Hibernate.initialize(course.getCourseType());
            session.getTransaction().commit();
        } catch (HibernateException e) {
            logger.error(e.getMessage());
        } finally {
            try {
                session.close();
            } catch (HibernateException e) {
                logger.error(e.getMessage());
            }
        }
        return course;
    }
}
```

You see how we only need to override the methods where the lazy fetch is "too lazy", where we need to initialize a child.

4.2 Jersey Services

Jersey is the reference implementation for the JSR 311 specification.

The Jersey implementation provides a library to implement Restful webservices in our Java servlet container. On the server side Jersey provides the servlet implementation which scans predefined classes to identify RESTful resources. In our web.xml configuration file we registered this servlet for our web application.

our xml file:

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <display-name>OLP Student Handler</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>jersey</servlet-name>
        <servlet-
class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>jersey</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
    <distributable/>
</web-app>
```

As you can see, the path is defined in the servlet-mapping right after the jersey servlet implementation. This implementation also provides a client library to communicate with the RESTful webservice, which base URL is:

```
http://localhost:8080/OlpStudentHandler/rest/class_name/type_of_return/
```

Where the type of return can be either json, plain, or xml. Here's an example with the Course rest service :

CourseService.java

```
@Path("/course")
public class CourseService {

    @Context
    private ServletContext sctx;          // dependency injection
    private static CourseDao dao;

    public CourseService() {
    }

    @GET
    @Produces({MediaType.APPLICATION_JSON})
    @Path("/json")
    public Response getJson() {
        checkContext();
        Set<CoursePersistance> set = dao.getAll();
        Set<Course> courses = new HashSet<>();
        for(CoursePersistance p:set){
            courses.add(new Course(p));
        }
        return Response.ok(toJson(courses), "application/json").build();
    }
```

And to trigger the getJson method, we use the following url :

```
http://localhost:8080/OlpStudentHandler/rest/course/json
```

To get the following answer :

```
[{"courseId":2,"contentProvider":
{"contentProviderId":1,"contentProviderName":"Lina","contentProviderDescription
":"Speciality : MS
office","contentProviderEmail":"lina@nerdson.dk"},"courseBranch":
{"courseBranchId":2,"courseBranchName":"Network"},"courseType":
{"courseTypeId":2,"courseBranchId":2,"courseTypeName":"Project
Management"},"courseName":"Gant diagram","courseDescription":"blah
blah","released":null,"language":"english","skillLevel":"intermediate"},
{"courseId":1,"contentProvider":
{"contentProviderId":1,"contentProviderName":"Lina","contentProviderDescription
":"Speciality : MS
office","contentProviderEmail":"lina@nerdson.dk"},"courseBranch":
{"courseBranchId":1,"courseBranchName":"IT"},"courseType":
{"courseTypeId":1,"courseBranchId":1,"courseTypeName":"Mobile
apps"},"courseName":"Android","courseDescription":"blah","released":null,"langu
age":"english","skillLevel":"beginner"}]
```

We then implemented most of the basic CRUD operation over all of our entities according to the application requirements.

course create json example:

```
    @POST
    @Produces({MediaType.APPLICATION_JSON})
    @Path("/json/create")
    public Response createJson(@QueryParam("course_name") String course_name,
            @QueryParam("description") String description,
            @QueryParam("skill_level") String skill_level,
            @QueryParam("language") String language) {
        checkContext();
        // Require name to create.
        if (course_name == null) {
            String msg = "Property 'course_name' is missing.\n";
            return Response.status(Response.Status.BAD_REQUEST).
                    entity(msg).
                    type(MediaType.APPLICATION_JSON).
                    build();
        }
        // Otherwise, create the Mail and add it to the database.
        CoursePersistance coursePersistance = new CoursePersistance();
        coursePersistance.setCourseName(course_name);
        coursePersistance.setCourseDescription(description);
        coursePersistance.setLanguage(language);
        coursePersistance.setSkillLevel(skill_level);
        dao.save(coursePersistance);
        Course course = new Course(coursePersistance);
        return Response.ok(toJson(course), MediaType.APPLICATION_JSON).build();
    }
```

course delete plain example:

```
    @DELETE
    @Produces({MediaType.TEXT_PLAIN})
    @Path("/plain/delete/{id: \\d+}")
    public Response delete(@PathParam("id") int id) {
        checkContext();
        String msg = null;
        CoursePersistance course = dao.findByID(id);
        if (course == null) {
            msg = "There is no course with id " + id + ". Cannot delete.\n";
            return Response.status(Response.Status.BAD_REQUEST).
                    entity(msg).
                    type(MediaType.TEXT_PLAIN).
                    build();
        }
        dao.delete(course);
        msg = "Course " + id + " deleted.\n";
        return Response.ok(msg, "text/plain").build();
    }
```

course put json example:

```
    @PUT
    @Produces({MediaType.APPLICATION_JSON})
    @Path("json/update/{id: \\d+}")
    public Response updateJson(@QueryParam("id") int id,
            @QueryParam("course_name") String course_name,
            @QueryParam("description") String description,
            @QueryParam("skill_level") String skill_level,
            @QueryParam("language") String language) {
        checkContext();

        System.out.println("in put json got request for id " + id);
        CoursePersistance course = dao.findByID(id);
        // Check that sufficient data are present to do an edit.
        String msg = null;
        if (course_name == null && description == null && skill_level == null
&& language == null) {
            msg = "No parameters is given: nothing to edit\n";
        } else if (course == null) {
            msg = "There is no course with id " + id + "\n";
        }

        if (msg != null) {
            return Response.status(Response.Status.BAD_REQUEST).
                    entity(msg).
                    type(MediaType.APPLICATION_JSON).
                    build();
        } else {
            // Update.
            course.setCourseName(course_name);
            course.setCourseDescription(description);
            course.setLanguage(language);
            course.setSkillLevel(skill_level);
        }
        dao.update(course);

        return Response.ok(toJson(new Course(course)),
MediaType.APPLICATION_JSON).build();
    }
```

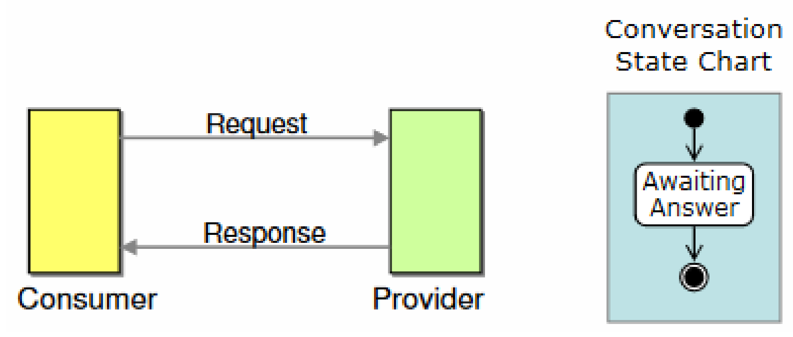
5. Integration patterns

Web service-oriented architectures put interaction into the spotlight and will most of the time implement lots of different integration patterns.

5.1 Conversation patterns

A conversation pattern provide a catalog of common scenarios and offer integration solutions on specific design problems, but the good thing is that they put focus on those particular design intents and trade-offs.

A. Request-Reply



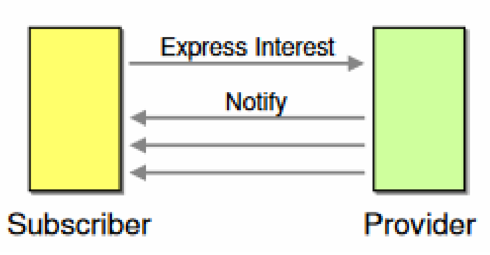
Request-Reply

• Single Conversation state: waiting for reply, complete

• More complicated once error conditions considered

This is a very simple interaction between two systems, which mimics a procedure call from one system to the other: here the client sends a request to the rest service, expecting a response message in return. The message-based interaction uses separate request and response messages.

B. Subscribe-Notify



Subscribe-Notify

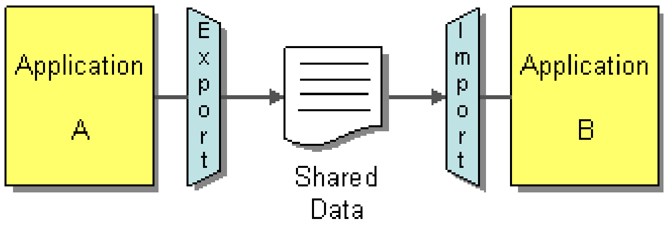
• Subscriber expresses interest in receiving notifications

• Subscriber receives messages until a stop condition is reached : sends a stop request

Subscribe-Notify assumes that the originator can receive inbound messages. It makes more efficient use of network resources than repeated Request-Response. Not counting administrative messages, such as renewals, Subscribe-Notify transmits only half as many messages as a repeated Request-Response. This is perfectly illustrated when for exemple a user wants to follow a course in our system.

5.2 Integration Styles

A. File Transfer



File Transfer

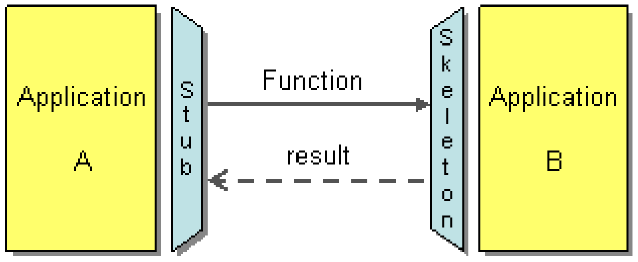
• Have each application produce files containing information that other applications need to consume.

• Integrators take the responsibility of transforming files into different formats.

• Produce the files at regular intervals according to the nature of the business

An important decision with files is what format to use. Which is why we chose XML, Json and plain text files to communicate between client and server.

B. Remote Procedure Invocation



Remote Procedure Invocation

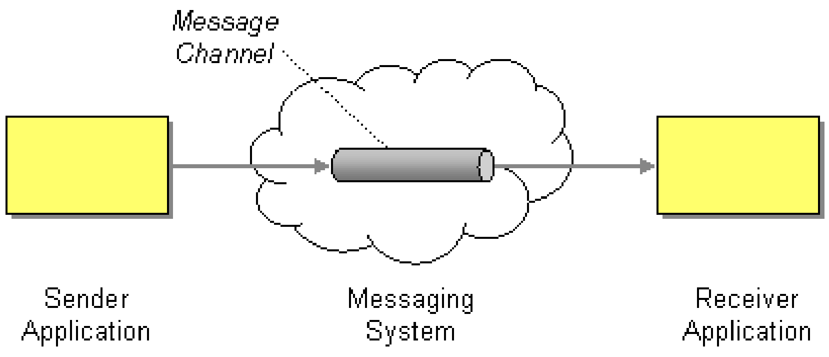
• Develop each application as a large-scale object or component with encapsulated data.

• Provide an interface to allow other applications to interact with the running application.

Remote Procedure Invocation applies the principle of encapsulation to integrating applications. If an application needs some information that is owned by another application, it asks that application directly. If one application needs to modify the data of another, then it does so by making a call to the other application. In our application this call is made through the REST service calling DAO methods.

5.3 Messaging system

A. Message Chanel



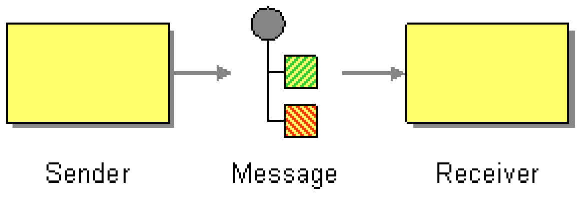
Message Chanel

• Connect the applications using a Message Channel

• Select a sender and a reciever

When an application has information to communicate, it doesn't just fling the information into the messaging system, it adds the information to a particular Message Channel. An application receiving information doesn't just pick it up at random from the messaging system; it retrieves the information from a particular Message Channel. The architechture we used uses the server as sender and the client as receiver. Both use the same REST channel : URI.

B. Message



Message

• Package the information into a Message, a data record that the messaging system can transmit through a message channel.

Thus any data that is to be transmitted via a messaging system must be converted into one or more messages that can be sent through messaging channels.

6. Conclusion

To conclude this report, we would say we were successful in the application of system integration concepts to our web service application. At the time of this writing, the project does not entirely fulfill our own requirements for what the application should do, but there are no technical barriers to fulfilling them. It has simply been due to a shortage of available development time. We are confident that, given another sprint's worth of time, we would have implemented all of the desired functionalities. The technologies we have been using so far have been really interesting to work with and were successful in showing off the main points of this course. Web services proved themselves great at exposing software functionality to customers and integrating heterogeneous platforms using open and commonly accepted Internet protocols.

The insights we have gained and shared in this report, section by section, helped us in our understanding of system design and integration patterns, and the lessons learned will aid us in our future iterations on similar application.

7. Appendices

7.1 References

RedHat *Hibernate* Retrieved Retrieved 05 13, 2016, from http://hibernate.org/orm/

Martin Fowler (8 May 2012) *OrmHate* Retrieved 05 13, 2016, from http://martinfowler.com/bliki/OrmHate.html

Gregor Hohpe (18 June 2008) *EIP* Retrieved 29 05, 2016 from http://www.enterpriseintegrationpatterns.com/