Student eXchange Process Modelling and Implementation by Using an Integrated BMP-SOA Approach

Octavian DOSPINESCU, Cătălin STRÎMBEI, Roxana-Marina STRAINU, Alexandra NISTOR
Faculty of Economics and Business Administration, AL.I.Cuza University, Iasi
doctav@uaic.ro, linus@uaic.ro, roxana.strainu@gmail.com, alexandra.anichitoaei@yahoo.com

One of the key processes of an open University Information System concerns managing the student exchange activities. In this paper we will try to address the challenges regarding modelling and implementation when integrating such a process by crossing different information systems. Our approach will leverage SOA architecture by using BPM in order to structure and build the service orchestration level.
Keywords: *BPM, SOA, JAX-RS, Service Oriented Architecture, RESTful Web Services*

1 BMP-SOA Integrated Approach

In a previous paper [1] we proposed an integrating methodology, briefly exposed in figure 1, starting from some specific methodologies that aim to bring in the same context the SOA architecture and BMP methodologies (or reverse BMP methodologies in the context of the SOA architectures) like SOAML [2], SOMF [2] or SOMA [3].

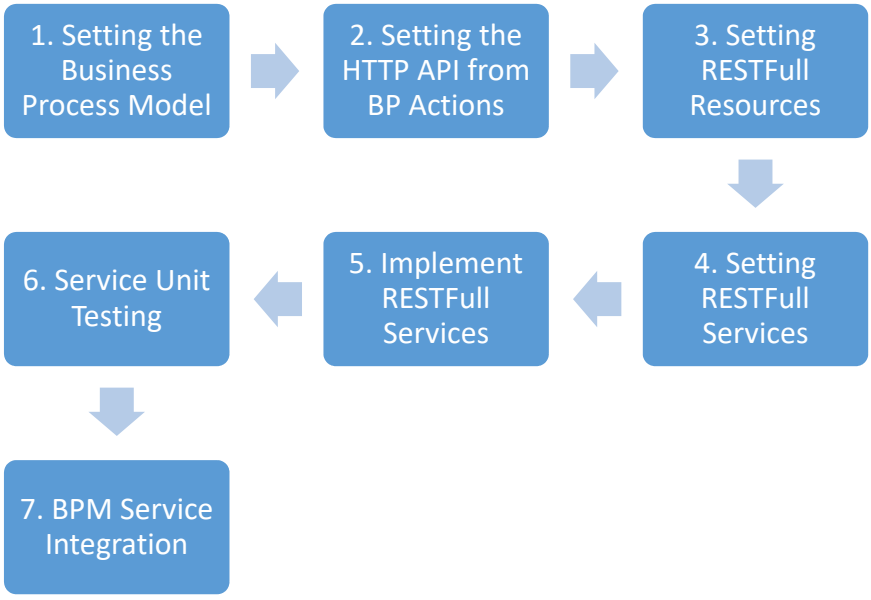


Fig. 1. BPM-SOA Proposal workflow

According to [4], [5], there are many ways to integrate different systems. Our approach tries to gradually transform BPM specifications in RESTfull specifications in order to be implemented using some common SOA frameworks like Java's JAX-RS, going through a set of distinct stages:

1. *Setting the Business Process (BP) Model* where each BP Action should be described using design specifications that will cover: the action identifier, action type (UIX, Atomic Processing, Synchronous, Timing), action data inputs and action data output.
2. *Setting the HTTP API from BP Action Specifications* where each BP Action will

DOI: 10.12948/issn14531305/22.1.2018.04

result into a HTTP Action that will be fully specified through: HTTP Action identifier: URI (from a base URL), HTTP Action Type (CRUD, RPC, UI/UX, Event based Asynchronous Acknowledgement) and HTTP Predicate, HTTP Action Input (meaning Input URL Parameters, Input Request headers, Request Body format: XML or JSON), HTTP Action Output (meaning Output Header: key-value result set, Response Body format, Response Code).

3. *Setting RESTFull Resources as a model of business entities* that will make the transition from process actions to an actually business data model by using HTTP CRUD Action types and identifying the underlying RESTfull resources.
4. *Setting RESTFull Services to provide RESTfull resources* (or business entities-based model) aiming to produce the modularization perspective to be used by the implementation of the underlying software components exposed as RESTfull services.
5. *Implementation of RESTFull Services*, using platforms like JAX-RS, Spring MVC, etc.
6. *Service Unit Testing* (service-level testing) where each RESTFull Service has to be deployed and "to live" into an autonomous executable context/runtime that will allow its validation by some modular and unit tests.
7. *BPM Service Integration and Testing* where service components will be integrated and orchestrated within a BP Platform Runtime (like jBPM, Bonita or Activity platforms) from where the initial Business Process could be executed and validated by integration tests.

2. Context of Student eXchange Process

In the following pages we will try to validate our BMP-to-SOA approach by implementing the above mapping guidelines into the practical context of the specific business process, targeting the integration of university information systems to support student exchange programs. This business process we have previously investigated in the larger context of University Information Systems [6].

We also take in consideration that according to [7], the educational offers must face the new challenges that require flexibility, rapidity, complexity and provide students both with specific habits and efficient work tools. In order to define a concrete context for our business process, we will describe the BMP entities or actors responsible for the BP actions to be mapped by using HTTP services.

BPM Entities/Actors

Table 1. Main actors from integrated systems

System	SubSystem/Service	URL
Partner University	SRMP	./part.univ/SRMP ./part.univ/SRMP/students ./part.univ/SRMP/grades
Origin University	SRMP	./origin.univ/SRMP ./origin.univ/SRMP/students ./origin.univ/SRMP/grades
Partner University	SPC	./part.univ/SPC ./part.univ/SPC/spec/disciplines
Origin University	SPC	./origin.univ/SPC

DOI: 10.12948/issn14531305/22.1.2018.04

		./origin.univ/SPC/spec/disciplines
Exchange Program (Erasmus)	SRC	./exchange/curriculum ./exchange/curriculum/equivalence
Exchange Program (Erasmus)	SRMP	./exchange/students ./exchange/grades/equivalence
Partner University	DOCX	

The definitions for the proposed terms are as following:

- SRMP means Student RoadMaP: Professor, curriculum, study programs, modules, timetable;
- SPC means Study Programs & Curriculum: Student, grades, disciplines, tests, location, time;
- DOCX refers to Students, professors, secretary, documents and announcements.

BPM Activities

Our simplified BP model proposed for student exchange programs is described in figure 2.

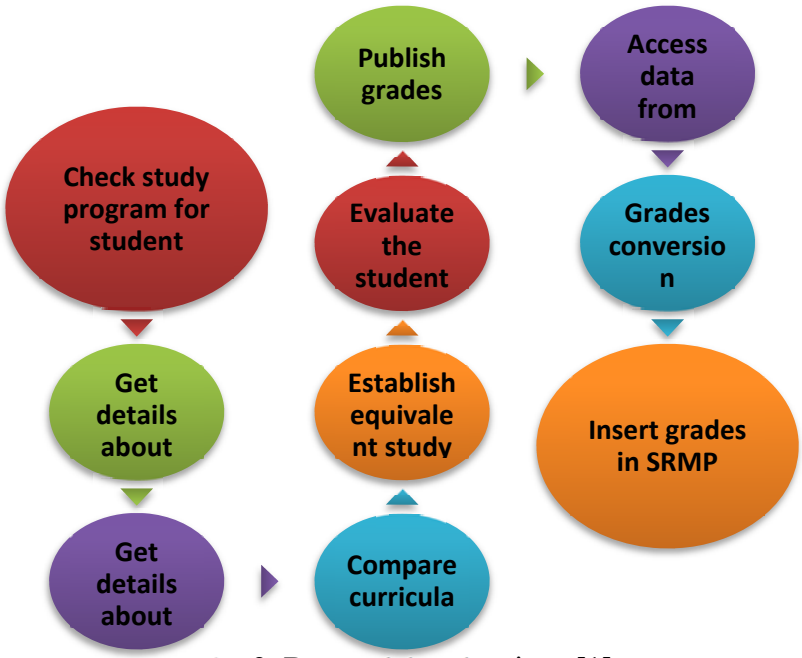


Fig. 2. BP Model and actions [1]

One could easily see that the BP model for student exchange programs tries to cover those activities related not only to student registration to another university information system, but also with importing student scholar data back to original university. This cross information cycle involves in fact a challenge regarding information integration. This challenge proved to be a complex issue for any university that accesses student-exchange academic programs.

3. Modelling Student eXchange Process with BMP and implementation in SOA context

Starting from the ideas described by the specialized literature [8], [9], [10], [11], in the following sections we propose a set of referential specifications that anyone could use as a reusable template easily adaptable in

DOI: 10.12948/issn14531305/22.1.2018.04

order to produce a fully operational service architecture in the domain of student exchange program management.

3.1 BPM Action Specification

We start by formalizing those specific business process action features that will determine the actual service boundaries and parameters.

First actions concern basic READ operations to get necessary information about study programs and course details from the partner university to establish equivalences.

Table 2. Action 1: <With an identification number of the student, identify the courses list for the student>, Action 2: <Identify details about the courses that the student attends>, Action 3: <Get details about the courses from the partner university>, Action 4: <Compare courses details to identify the courses the student will have to attend at the partner university>

	Action 1	Action 2	Action 3	Action 4
Action Name	Check study program for student	Get details about student courses	Get details about the courses	Compare curricula
Action Type	READ	READ	READ	READ
Action Data Input	Student ID	Courses List from SRMP	Year and semester of study	Courses details from origin and partner universities
Action Data Output	Courses List from SRMP	Courses Details from SPC	Courses details from SPC	Equivalent courses

Next actions concern basic transactions necessary to acquire selected courses from the partner university.

Table 3. Action 5: <Decide which classes a student will have to attend to have a match in the SRMP>, Action 6: <Student evaluation which will take place in the partner university>

	Action 5	Action 6
Action Name	Establish equivalent study courses	Evaluate the student (SRMP)
Action Type	WRITE	WRITE
Action Data Input	Equivalent courses	Courses to attend, student ID, grades
Action Data Output	Courses to attend	Grades from Partner University

Last actions are about exporting student grades from the partner university to students' original university.

DOI: 10.12948/issn14531305/22.1.2018.04

Table 4. Action 7: <The grades must be published to be accessed from inside and outside>, Action 8: <After the student is evaluated the grades from Partner University will be accessed>, Action 9:<After the student is evaluated the grades will be converted in different grading systems>, Action 10: <Final Grades are inserted in parent university database >

	Action 7	Action 8	Action 9	Action 10
Action Name	Publish grades	Access data from partner university	Grades conversion	Insert grades in SRMP
Action Type	READ	READ	UPDATE	WRITE
Action Data Input	Student ID, grades and converted grades from Partner University	Credentials, student ID	Student ID, Grades from partner university	XML/JSON Final Grades Data
Action Data Output	Grades Data	Student grades (XML/JSON)	Converted grades	Service (message with status)

3.2 HTTP Action Specifications [HTTP API]

By mapping business actions from initial BP model to HTTP predicates will result a new set of specific HTTP Actions.

As in previous section, we first present actions and formalized HTTP operations concerning identification of eligible courses from the partner university.

Table 5. Action HTTP 1: < Identify the courses list for the student>, Action HTTP 2: < Identify details about the courses that the student attends >, Action HTTP 3: < Get details about the courses from the partner university>>, Action HTTP 4: < Compare courses details to identify the courses the student will have to attend at the partner university >

	HTTP Action 1	HTTP Action 2	HTTP Action 3	HTTP Action 4
HTTP Action URL	http://server:host/SRS/<sub_module>/.../parent.univ/STX/students/studentID/course	http://server:host/SRS/<sub_module>/…/parent.univ/STX/courses/speciality/specName/courselist	http://server:host/SRS/<sub_module>/.../part.univ/STX/courses/mainfield	http://server:host/SRS/<sub_module>/..../part.univ/STX/courses/mainfield/coursefield
BPM Action Name	Check study program for student	Search Student Details at parent university	Get details about the courses	Compare curricula
HTTP Action Type	READ	READ	READ	READ
HTTP	GET	GET	GET	GET

DOI: 10.12948/issn14531305/22.1.2018.04

Predicate				
[Input] URL Parameters		semester	semester	semester
[Input] Request Body				
[Output] HTTP Response Code	200	200	200	200
[Output] Response Body	Courses List from SRMP (XML/JSON)	Courses details (XML/JSON)	Courses details (XML/Json)	Equivalent courses (XML/Json)

Next actions formalize the necessary transactions to the partner university. operations to define student enrollment

Table 6. Action HTTP 5: < Decide which classes a student will have to attend to have a match in the SRMP >, Action HTTP 6: < Student evaluation which will take place in the partner university >

	HTTP Action 5	HTTP Action 6
HTTP Action URL	http://server:host/SRS/<sub_module>.../parent.univ/STX/equalizations	http://server:host/SRS/<sub_module>/…./part.univ/STX/grades/studentId
BPM Action Name	Access parent university service	Evaluate the student (SRMP)
HTTP Action Type	CREATE	CREATE
HTTP Predicate	POST	POST
[Input] URL Parameters		
[Input] Request Body	{ "convertedScore": "", "courseName": "Logical games", "eqCourseName": "Logic&design", "eqId": , "score": "", "semester": 2, "studentId": "student2", "year": 2016 }	{ "course": "Logic&design", "grade": "C", "id": 5, "needsConversion": true, "scoringSystem": "swedish", "semester": 2, "studentID": "student2", "year": 2016 }
[Output] HTTP Response	200	200

DOI: 10.12948/issn14531305/22.1.2018.04

Code		
[Output] Response Body	Courses to attend,semester,studentID, year (JSON)	Grades from Partner University (XML/JSON)

Finally, last HTTP actions formalize the grades import transactions from the partner university information system to the information system of students' original university.

Table 7. Action HTTP 7: < The grades must be published to be accessed from inside and outside >, Action HTTP 8: < After the student is evaluated the grades from Partner University will be accessed>, Action HTTP 9: < After the student is evaluated the grades will be converted in different grading systems and updated into equalization system >, Action HTTP 10: <Final Grades are inserted into parent university database>

	HTTP Action 7	HTTP Action 8	HTTP Action 9	HTTP Action 10
HTTP Action URL	http://server:host/SRS/<sub_module>/.../part.univ/STX/grades/studentId	http://server:host/SRS/<sub_module>/.../part.univ/STX/grades/studentID	http://server:host/SRS/<sub_module>/..../parent.univ/STX/equalizations	http://server:host/SRS/<sub_module>/.../parent.univ/STX/grades
BPM Action Name	Publish grades	Access data from partner university	Student ID, Grades from partner university	Insert Grades into parent university database
HTTP Action Type	READ	READ	UPDATE	CREATE
HTTP Predicate	GET	GET	PUT	POST
[Input] URL Parameters			Year,semester,studentID,eqCourseName	
[Input] Request Body			{ "convertedScore": "7", "courseName": "Logical games", "eqCourseName": "Logic&design", "eqId": 1, "score":	{ "course": "Logical games", "dateGranted": "2016-07-09", "grade": "7", "id": auto, "needsConversion": false,

DOI: 10.12948/issn14531305/22.1.2018.04

			"C", "semester": 2, "studentId": "student2", "year": 2016 }	"scoringSystem": "romanian", "semester": 2, "studentID": "student2", "year": 2016 }
[Output] HTTP Response Code	200	200	200	200
[Output] Response Body	Grades Data (XML/JSON)	Student grades (XML/JSON)	Converted grades (XML/JSON)	Service (message with status)

3.3 Implementation approach of Student Exchange REST model

The structure of our business data model is designed in a way to conform to the BPM requirements of the project. To accomplish this, we needed:

- a model of entity classes which are the equivalent of the tables from a database;
- a repository class to manage data queries from the database using model classes;
- service classes use data from database and apply specific methods for lists of data, to implement specific operations for REST resources;
- resource classes which contain instances of services and the REST infrastructure.

The implementation context used refers to JEE platform with JPA-ORM framework (Hibernate), JAX-RS using Jersey implementation and JAXB-OXM (Object to XML/JSON mapping) with Jackson Provider.

3.3.1 Specifications of REST Resource Model

The hierarchic implementation of classes is as it may be seen in figure 3 below:

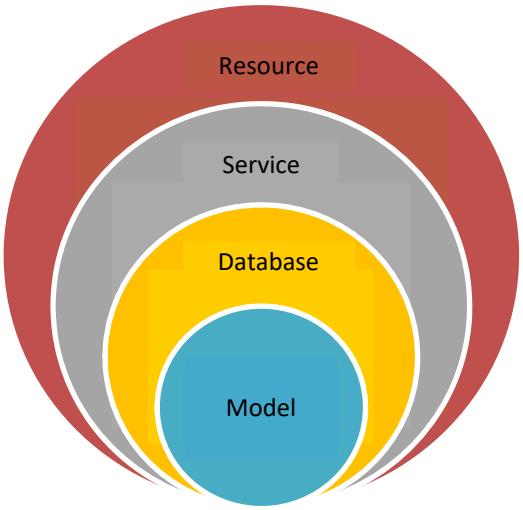


Fig. 3. Hierarchic implementation of model specs

The (data) *model classes* (located in model package within the project) are: *Course, Student, Grade, Equalization* Each instance of these classes represents a record into the corresponding database table. Using another class (DatabaseClass) we extract the data from the database using lists of each model class. These classes are simple implementations of Java-Beans conventions as shown in figure 4.

DOI: 10.12948/issn14531305/22.1.2018.04

Fig. 4. Course Model Class

Listing 1. Course model class with XML-OXM annotations (OXM: Object-to-XML/JSON)

```
@XmlRootElement
public class Course {
	private long courseID;
	private String courseName;
	private String spec;
	private Integer semester;
	private String mainField;
	private String courseField;
… … …
// Getters and setters java-bean methods
}
```

The *service classes* (located in services package within the project) are as follows:

- *CourseService*: offers the possibility to view all courses, to view a course by id, to get courses by speciality, by main field (the domain of the course), by course field (a branch from the domain), by semester, to delete, add or update a course.

Listing 2. Course Service implementation

```
public List<Course> getAllCourses(){
	return new ArrayList<Course>(courses.values());
    }

    public Course getCourse(long courseId){
        return courses.get(courseId);
    }

    public List<Course> getCoursesBySpeciality(String spec){
        List<Course> coursesBySpec= new ArrayList<Course>();
        for(Course crs:courses.values()){
               if(crs.getSpec().toLowerCase().equals(spec.toLowerCase())){
                     coursesBySpec.add(crs);
               }
```

DOI: 10.12948/issn14531305/22.1.2018.04

```
        }
        return coursesBySpec;
    }

    public List<Course> getCoursesByMainField(String mainField){
        List<Course> coursesByField = new ArrayList<Course>();
        for(Course crs:courses.values()){
               if(crs.getMainField().toLowerCase().equals(mainField.toLowerCase())){
                      coursesByField.add(crs);
               }
        }
        return coursesByField;
    }

    public List<Course> getCoursesByCourseField(String courseField){
        List<Course> coursesByField = new ArrayList<Course>();
        for(Course crs:courses.values()){

if(crs.getCourseField().toLowerCase().equals(courseField.toLowerCase())){
                      coursesByField.add(crs);
               }
        }
        return coursesByField;
    }

    public List<Course> getCoursesBySemester(String spec,int semester){
        List<Course> coursesBySem = new ArrayList<Course>();
        for(Course crs:courses.values()){
               if(crs.getSpec().toLowerCase().equals(spec.toLowerCase())){
                      if(crs.getSemester()==semester){
                             coursesBySem.add(crs);
                      }
               }
        }
        return coursesBySem;
    }

    public Course addCourse(Course course){
        course.setCourseID(courses.size()+1);
        courses.put(course.getCourseID(),course);
        return course;
    }
    public Course updateCourse(Course course){
        if(course.getCourseID()<=0){
               return null;
        }
        courses.put(course.getCourseID(), course);
        return course;
    }
    public Course removeCourse(long courseID){
        return courses.remove(courseID);
    }
```

- *StudentService*: offers the possibility to add, remove, update a student and get information about all students, students by specialties or information by one student by id.
- *GradeService* offers the possibility to add, remove, update a grade and to see the grades of a student, or by course, or by student and course name.
- *EqualizationService* offers the possibility to add, delete, update equalizations, to view all equalizations, to view equalizations by year and semester or by student.

Listing 3. CourseService method to get courses by filter

```
public Map<Long, Course> courses = DatabaseClass.getCourses();
```

DOI: 10.12948/issn14531305/22.1.2018.04

```
public List<Course> getCoursesBySpeciality(String spec){
             List<Course> coursesBySpec= new ArrayList<Course>();
             for(Course crs:courses.values()){
                    if(crs.getSpec().toLowerCase().equals(spec.toLowerCase())){
                           coursesBySpec.add(crs);
                    }
             }
             return coursesBySpec;
}
```

According to the specialized literature [12], the *resource classes* will be the ones that contain our REST architecture. The REST implementation of this model means that each method will have a @GET, @POST, @PUT, @DELETE (and @Path if it's the case) annotation attached, each class will have a @Path annotation attached, and the return type of the http request using @Produces and @Consumes annotations. The specific REST annotations are included in Jersey Library, in javax.ws.rs package. The annotation @PathParam is used to get data from the URL while @QueryParam is used to get data from URL parameters.

The *resource classes* are:

- *CourseResource* with /courses default path sets paths and specific actions over Course objects:
  /courses will give the list of all courses available
  /courses/mainfiled will give the list of courses filtered by mainfield, and it can return results using an url parameter to filter results by semester
  /courses/mainfield/coursefield will return a list of courses filtered by mainfield and coursefield, and it can return results using an URL parameter to filter results by semester (see figure 5).
  /courses/specialty/specname/courselist will return the list of courses for the specialty specname, and it can return results using an url parameter to filter results by semester (see figure 5).
  /courses with POST,PUT,DELETE actions will add, update or delete a Course object (in JSON format).

Listing 4. CourseResource class to produce Course JSON documents

```
@Path("/courses")

@Consumes({MediaType.APPLICATION_JSON})
@Produces(MediaType.APPLICATION_JSON)

public class CourseResource {
       CourseService courseService = new CourseService();
       @GET
       public List<Course> getCourses(){
              return courseService.getAllCourses();
       }
       @GET
       @Path("/speciality/{spec: .*}/courselist")
       public List<Course> getCoursesBySpec(@PathParam("spec") String spec,
            @QueryParam("semester") int sem){
              if(sem>0){
                     return courseService.getCoursesBySemester(spec,sem);
              }
              return courseService.getCoursesBySpeciality(spec);
       }
… … …
// other REST-HTTP mapping methods
}
```

DOI: 10.12948/issn14531305/22.1.2018.04

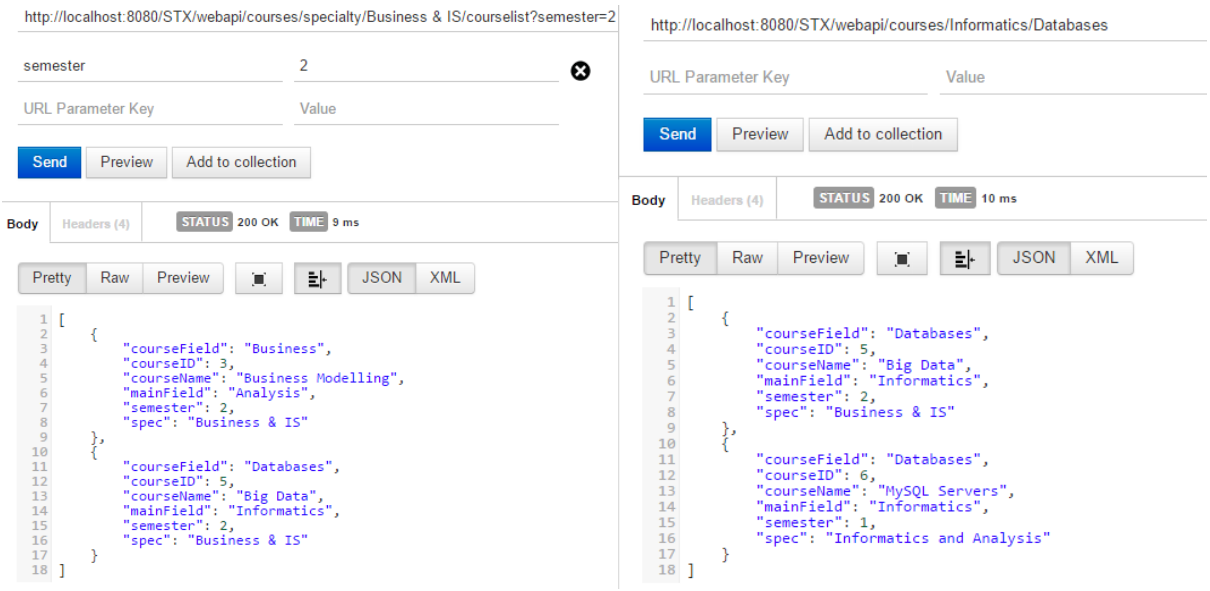


Fig. 5. Courses by specialty and semester (left) and by main field and course field (right)

- *StudentResource*: with /students as default path, sets paths and specific actions over Student objects:
  /students will return a list of all students
  /students/studentId will return the information about student with id studentId
  /students/studentId/grades will return the list of all grades for the student having id studentId (see figure 6).
  /students/studentId/courses will return the list of all courses for the student having id studentId and it can receive and url parameter to filter results by semester.
  /students/studentId/equalizations will return the list of all equalizations for the student having id studentId (see figure 6).
  /students with POST, PUT will update or add a Student object.
  /students/studentId with DELETE action will delete the student with id studentId.

DOI: 10.12948/issn14531305/22.1.2018.04

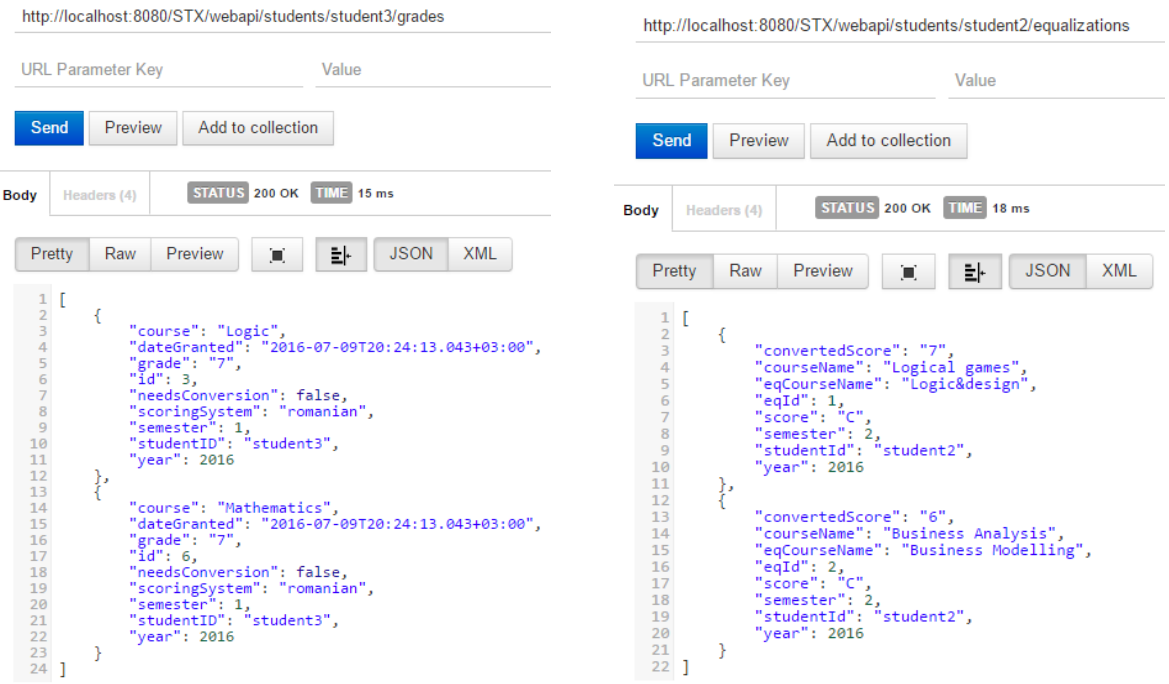


Fig. 6. Grades for student 3 (left) and equalizations for student 2 (right)

- *GradeResource* with /grades as the default path, sets paths and specific HTTP actions over Grade objects:
  - /grades will return a list of all grades
  - /grades/courses/courseName will return a list of grades for the course with name courseName (see Figure 7)
  - /grades/studentId will return the grades for the student with id studentId (see Figure 7)
  - /grades/studentId/equalizations will return the grades from equalization system for the student with id studentId
  - /grades with POST, PUT and DELETE actions will add, update or delete a Student object (in JSON format).

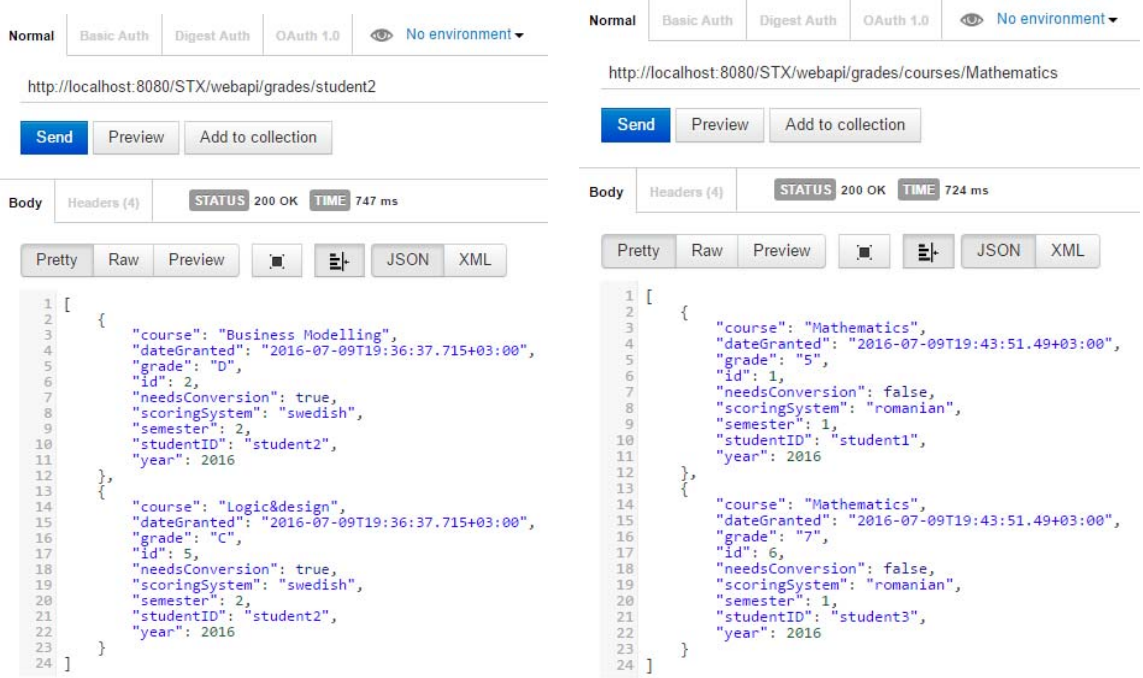


Fig. 7. Grades by student id (left) and by course name (right)

DOI: 10.12948/issn14531305/22.1.2018.04

- *EqualizationResource* with /equalizations as the default path, sets paths and specific HTTP actions over Equalization objects:
  /equalizations will return a list of all equalizations with the possibility to add URL parameters to filter results by year and semester (see figure 8).
  /equalizations/studentId will return a list of equalizations for the student with id studentId (see figure 8).
  /equalizations with POST and DELETE methods will delete the given equalization (in JSON format).
  /equalizations with PUT and URL parameters will update the equalization given by year, semester, studentId and equivalentCourseName with the given values for score and convertedScore.

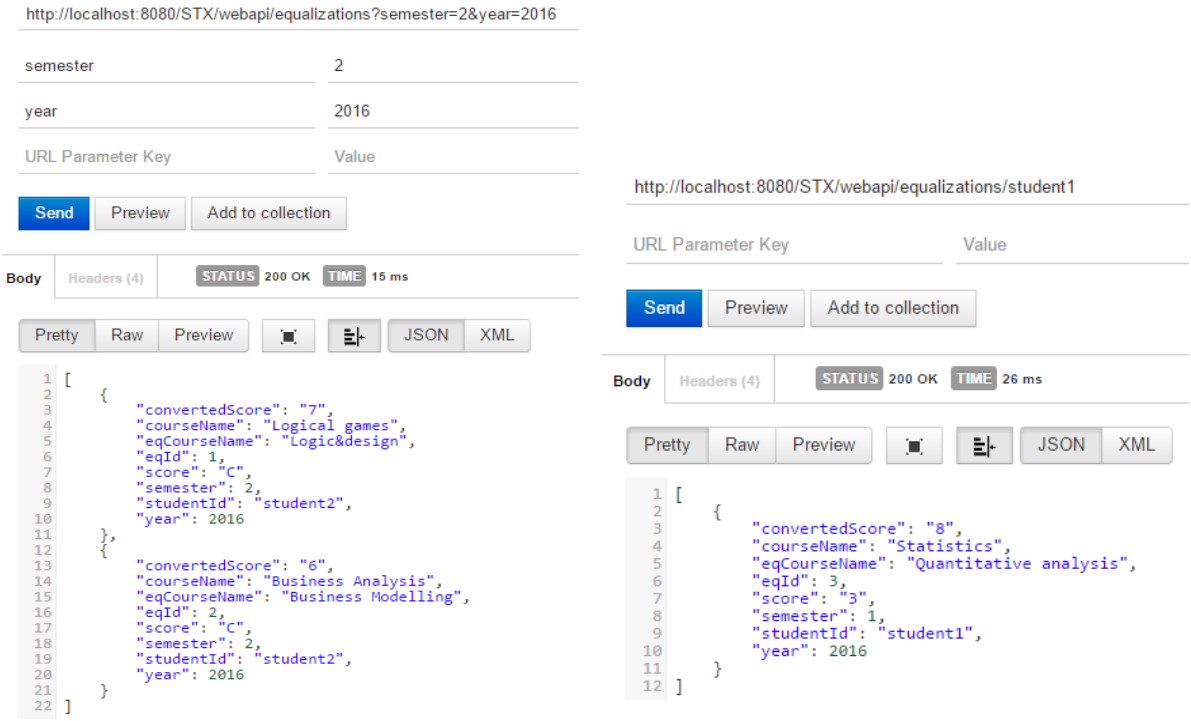


Fig. 8. Equalizations by year and semester (left) and for student1 (right)

4. Business Process Integration Model

Starting from the HTTP specification and implementation of Student Exchange REST model we made [1] the BPM.REST Action Model, which is based on a number of variables: Name, Request Body and Response Body. Following the specifications we create a project on jBMP platform (6.2.0 version) and we obtained the process diagram illustrated in figure 9.

DOI: 10.12948/issn14531305/22.1.2018.04

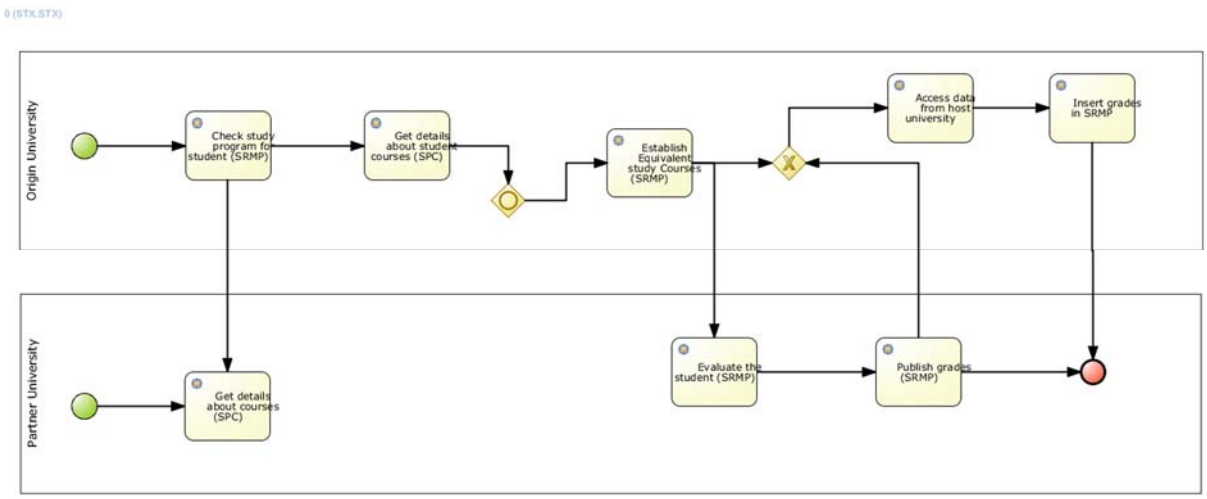


Fig. 9. BPM.REST Action Model (according to [1])

Each BPM.REST action have some parameters and parameters assignment. The common parameters are the URL of the action endpoint and for each HTTP method of request we have parameter named Method. Below we have illustrated the Action Parameters and Action Parameters Assignment under each Action.

BPM.REST Action 1 [Check study program for student(SRMP)] Specifications

- Action REST Resource Target (from REST Resource Model) - StudentResource
- Action Parameters and Parameters Assignment

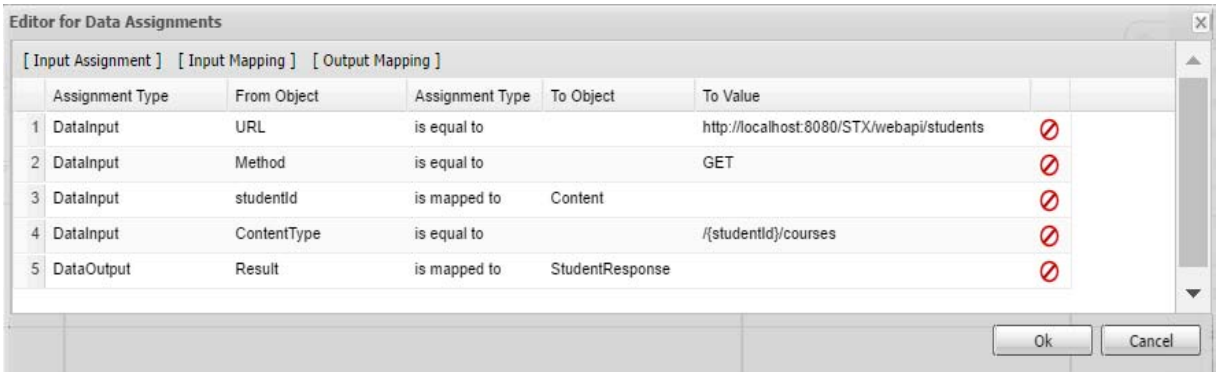


Fig. 10. Action 1 specs: Check study program for student

BPM.REST Action 2 [Get details about student courses] Specifications

- Action REST Resource Target (from REST Resource Model) - CourseResource

DOI: 10.12948/issn14531305/22.1.2018.04

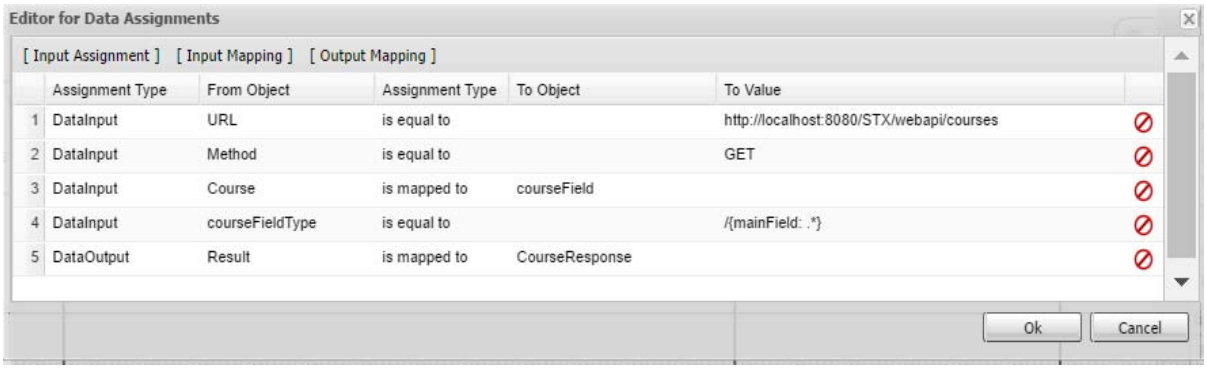


Fig. 11. Action 1 specs: Get details about student courses

BPM.REST Action 3 [Get details about courses (SPC)] Specifications

- Action REST Resource Target (from REST Resource Model) - CourseResource

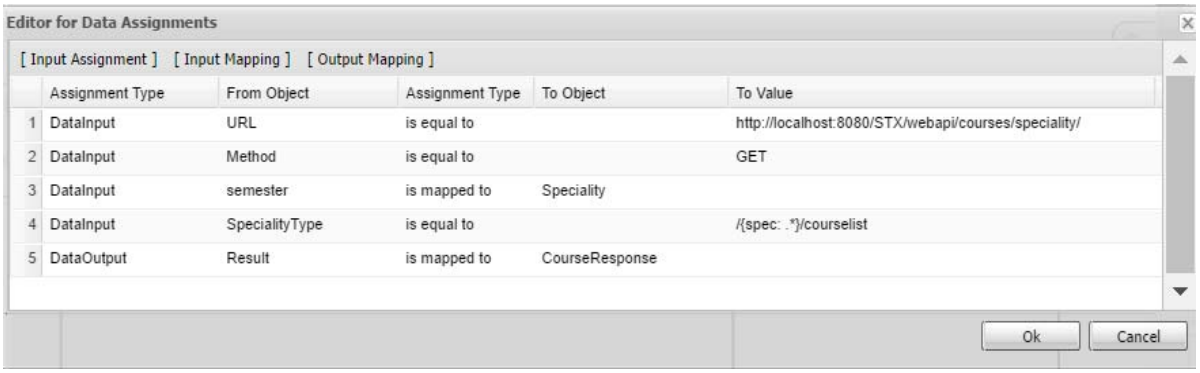


Fig. 12. Action 1 specs: Get details about courses

BPM.REST Action 4 [Establish Equivalent study Courses (SRMP)] Specifications

- Action REST Resource Target (from REST Resource Model) - StudentResource

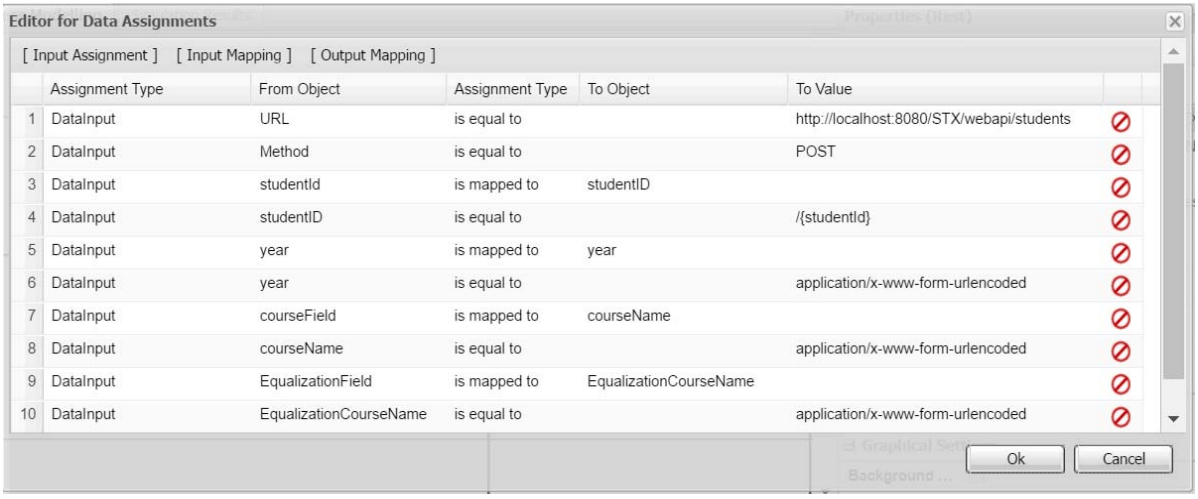
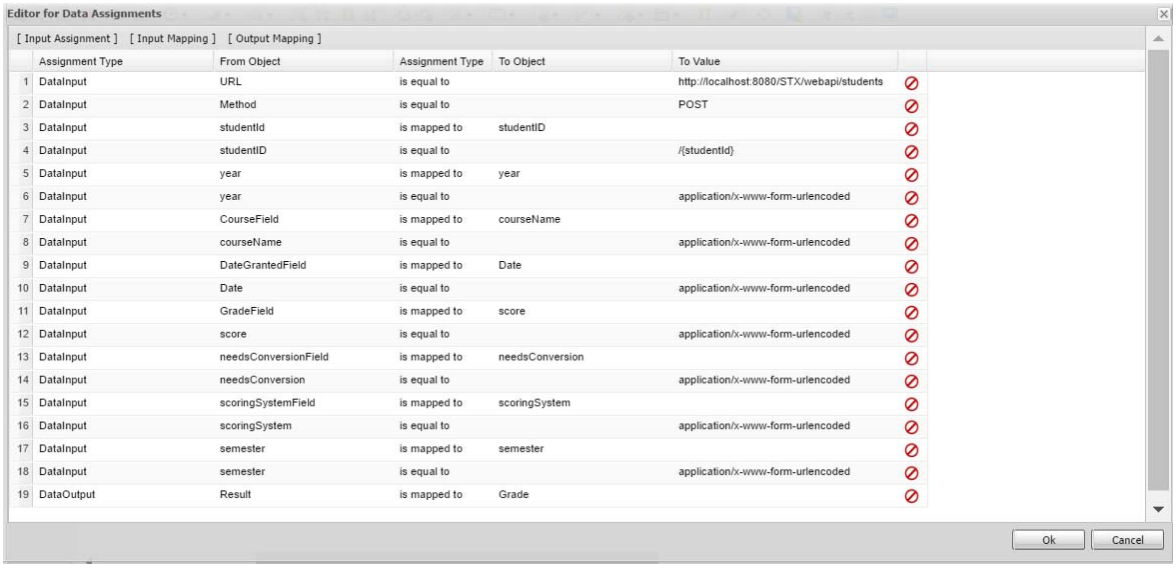


Fig. 13. Action 1 specs: Establish Equivalent study Courses

BPM.REST Action 5 [Evaluate the student (SRMP)] Specifications

- Action REST Resource Target (from REST Resource Model) - StudentResource

DOI: 10.12948/issn14531305/22.1.2018.04

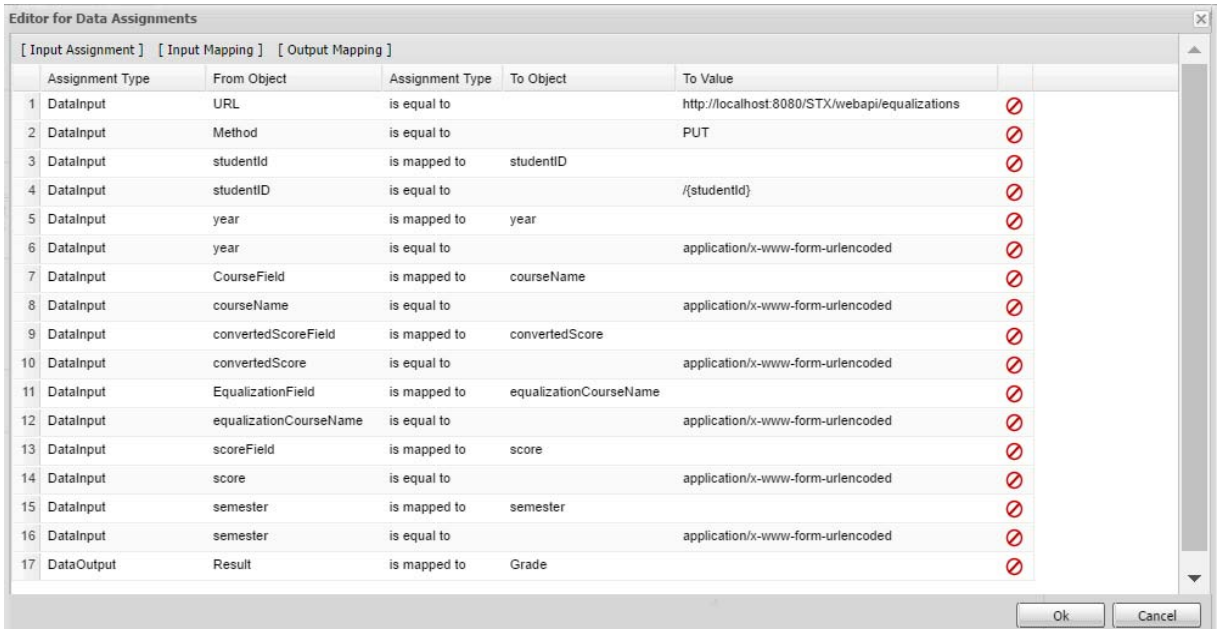


	Assignment Type	From Object	Assignment Type	To Object	To Value
1	DataInput	URL	is equal to		http://localhost:8080/STX/webapi/students
2	DataInput	Method	is equal to		POST
3	DataInput	studentId	is mapped to	studentID	
4	DataInput	studentID	is equal to		/{studentId}
5	DataInput	year	is mapped to	year	
6	DataInput	year	is equal to		application/x-www-form-urlencoded
7	DataInput	CourseField	is mapped to	courseName	
8	DataInput	courseName	is equal to		application/x-www-form-urlencoded
9	DataInput	DateGrantedField	is mapped to	Date	
10	DataInput	Date	is equal to		application/x-www-form-urlencoded
11	DataInput	GradeField	is mapped to	score	
12	DataInput	score	is equal to		application/x-www-form-urlencoded
13	DataInput	needsConversionField	is mapped to	needsConversion	
14	DataInput	needsConversion	is equal to		application/x-www-form-urlencoded
15	DataInput	scoringSystemField	is mapped to	scoringSystem	
16	DataInput	scoringSystem	is equal to		application/x-www-form-urlencoded
17	DataInput	semester	is mapped to	semester	
18	DataInput	semester	is equal to		application/x-www-form-urlencoded
19	DataOutput	Result	is mapped to	Grade	

Fig. 14. Action 1 specs: Evaluate the student

BPM.REST Action 6 [Publish grades (SRMP)] Specifications

- Action REST Resource Target (from REST Resource Model) - EqualizationResource



	Assignment Type	From Object	Assignment Type	To Object	To Value
1	DataInput	URL	is equal to		http://localhost:8080/STX/webapi/equalizations
2	DataInput	Method	is equal to		PUT
3	DataInput	studentId	is mapped to	studentID	
4	DataInput	studentID	is equal to		/{studentId}
5	DataInput	year	is mapped to	year	
6	DataInput	year	is equal to		application/x-www-form-urlencoded
7	DataInput	CourseField	is mapped to	courseName	
8	DataInput	courseName	is equal to		application/x-www-form-urlencoded
9	DataInput	convertedScoreField	is mapped to	convertedScore	
10	DataInput	convertedScore	is equal to		application/x-www-form-urlencoded
11	DataInput	EqualizationField	is mapped to	equalizationCourseName	
12	DataInput	equalizationCourseName	is equal to		application/x-www-form-urlencoded
13	DataInput	scoreField	is mapped to	score	
14	DataInput	score	is equal to		application/x-www-form-urlencoded
15	DataInput	semester	is mapped to	semester	
16	DataInput	semester	is equal to		application/x-www-form-urlencoded
17	DataOutput	Result	is mapped to	Grade	

Fig. 15. Action 1 specs: Publish grades

BPM.REST Action 7 [Access data from Host University] Specifications

- Action REST Resource Target (from REST Resource Model)

DOI: 10.12948/issn14531305/22.1.2018.04

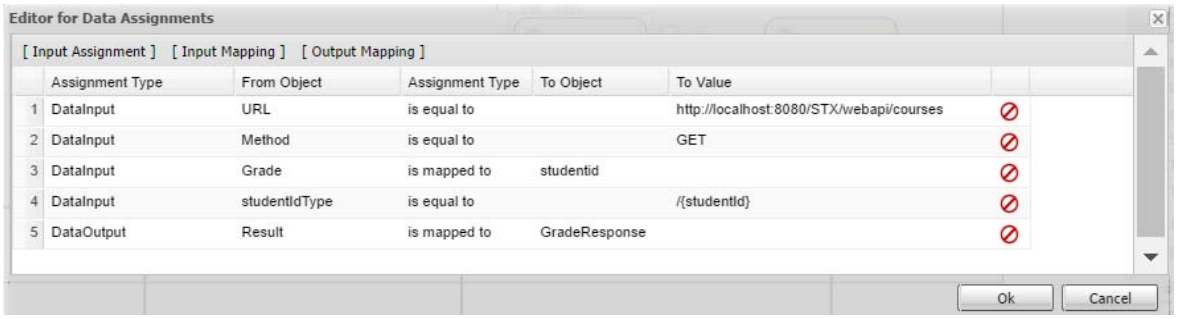


Fig. 16. Action 1 specs: Access data from Host University

BPM.REST Action 8 [Insert grades in SRMP] Specifications

- Action REST Resource Target (from REST Resource Model) - StudentResource

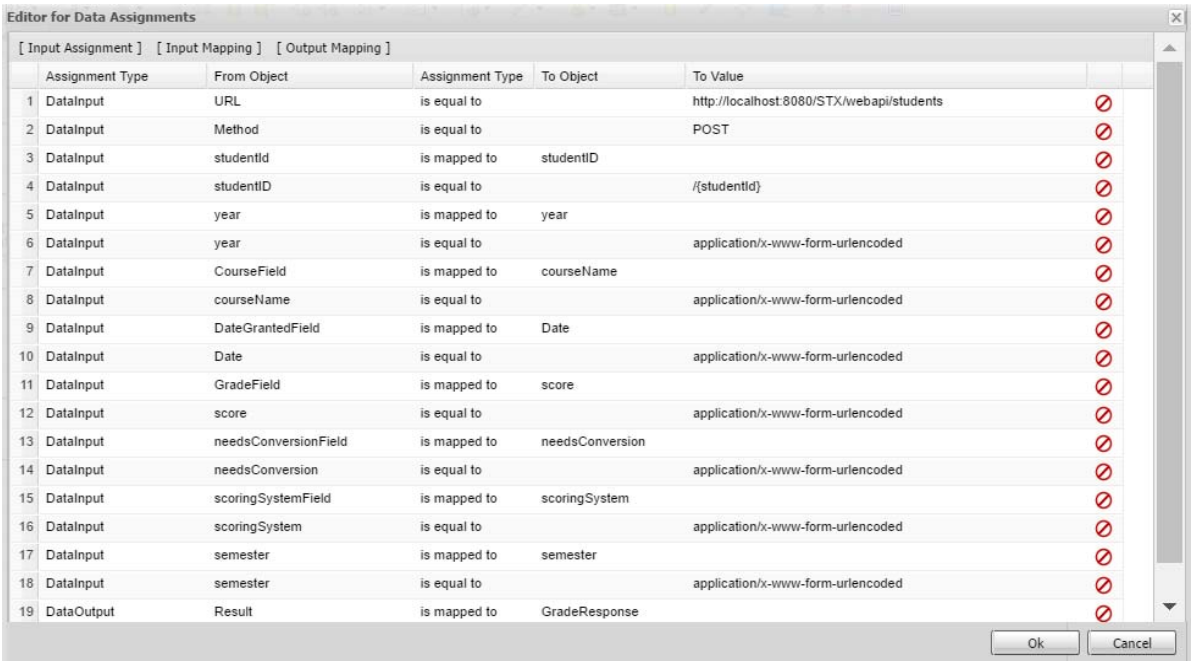


Fig. 17. Action 1 specs: Insert grades in SRMP

5. Conclusions

In this paper we have tried to achieve two goals. On the one hand, we made an extensive effort to build a complex of services that could automate the Student eXchange activities in a process that could be useful for those universities having students involved in international programs which are searching for a way to make these management processes more effective and transparent. On the other side, we have tried to make an experimental validation of our BPM-to-SOA modelling and to develop an approach in a relevant context inspired from a real (academic) problem encountered within actual University Information Systems.

Throughout our project we have tried to follow an end-to-end approach in order to cover the most relevant and critical aspects specific to SOA architectures. Our intentions were not to build a new SOA methodology, but to show a practical way on how to complement existing SOA approaches with the advantages of BPM methodologies, tools and platforms.

Although SOA and BMP methodologies emerged and evolved in parallel, we found that they could be fully compatible to build a mix between the very declarative approach of existing BPM platforms (meaning no code … just model, at one extreme) and the very customizable approach of SOA implementing

DOI: 10.12948/issn14531305/22.1.2018.04

platforms (assuming complex and proprietary integration and orchestration protocols, at the other extreme). We prove that declarative (even visually) orchestration is possible for service-based actions built in a customized and extensible manner.

Acknowledgments:
1. This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS – UEFISCDI, project number PN-II-RU-TE-2014-4-0748.
2. Some of the findings reported in this article were also orally presented at The Fourth Annual Conference on Global Higher Education at Tokyo held on June, 10th, 2017, at Lakeland University Japan, according to the official schedule available at: http://conference.lcjapan.com/schedule.html and http://conference.lcjapan.com/info.php?topic=34.

References

[1] O. Dospinescu, C. Strimbei, R. Strainu and A. Nistor, "REST SOA Orchestration and BPM Platforms," *Informatica Economica,* vol. 21, no. 1, pp. 30-42, 2017.

[2] M. Mohsen and M. Muriati, "A review of SOA Modeling Approaches for Enterprise Information Systems," *Procedia Technology,* vol. 11, pp. 794-800, 2013.

[3] IBM Business Consulting Services, "IBM Service-Oriented Modeling and Architecture," NY, 2004.

[4] C. Garcia and R. Abilio, "Systems Integration Using Web Services, REST and SOAP: A Practical Report," *Revista de Sistemas de Informação da FSMA,* vol. 1, no. 19, pp. 34-41, 2017.

[5] L. Burita and K. Zeman, "Architecture Approach in System Development," *Journal of Systems Integration,* vol. 8, no. 1, pp. 31-44, 2017.

[6] C. Strîmbei, O. Dospinescu, R. Strainu and A. Nistor, "The BPMN Approach of the University Information Systems," *Ecoforum Journal,* vol. 5, no. 2, pp. 181-193, 2016.

[7] N. Dospinescu, M. Tătăruşanu, G. Butnaru and L. Berechet, "The Perception of Students from the Economic Area on the New Learning Methods in the Knowledge Society," *The Amfiteatru Economic Journal,* vol. 13, no. 30, pp. 527-543, 2011.

[8] K. Stankevičius and O. Vasilecas, "Research On Rules-Based Business Process Modelling And Simulation," *SCIENCE - FUTURE OF LITHUANIA,* vol. 6, no. 2, pp. 147-150, 2014.

[9] M. Mohammadi and M. Mukhtar, "Business Process Modelling Languages in Designing Integrated Information System for Supply Chain Management," *International Journal on Advanced Science, Engineering and Information Technology,* vol. 2, no. 6, pp. 464-467, 2012.

[10] M. Mohammadi, "Combination of Modeling Techniques for Supporting Business Process Architecture Layers," *International Journal on Advanced Science, Engineering and Information Technology,* vol. 7, no. 3, pp. 1038-1048, 2017.

[11] M. Ahmadi and A. Nikravanshalmani, "Providing a framework to improve the performance of business process management projects based on BPMN," *Advances in Computer Science : an International Journal,* vol. 5, no. 1, pp. 10-17, 2016.

[12] S. Kennedy, O. Molloy, R. Stewart, P. Jacob, M. Maleshkova and F. Doheny, "A Semantically Automated Protocol Adapter for Mapping SOAP Web Services to RESTful HTTP Format to Enable the Web Infrastructure, Enhance Web Service Interoperability and Ease Web Service Migration," *Future Internet,* vol. 4, pp. 372-95, 2012.

DOI: 10.12948/issn14531305/22.1.2018.04

Octavian DOSPINESCU graduated the Faculty of Economics and Business Administration in 2000 and the Faculty of Informatics in 2001. He achieved the PhD in 2009 and he has published as author or co-author over 30 articles. He is author and co-author of 10 books and teaches as an associate professor in the Department of Information Systems of the Faculty of Economics and Business Administration, University Alexandru Ioan Cuza, Iasi. Since 2010 he has been a Microsoft Certified Professional, Dynamics Navision, Trade&Inventory Module. In 2014 he successfully completed the course "Programming Mobile Applications for Android Handheld Systems" authorized by Maryland University. He is interested in mobile devices software, computer programming and decision support systems.

Cătălin STRÎMBEI has graduated the Faculty of Economics and Business Administration of Al.I.Cuza University of Iaşi in 1997. He holds a PhD diploma in Cybernetics, Statistics and Business Informatics from 2006 and he has joined the staff of the Faculty of Economics and Business Administration as teaching assistant in 1998 and as associate professor in 2013. Currently he is teaching *Object Oriented Programming*, *Multi-Tier Software Application Development* and *Database Design and Administration* within the Department of Business Information Systems, Faculty of Economics and Business Administration, Al.I.Cuza University of Iaşi. He is the author and co-author of four books and over 30 journal articles in the field of object oriented development of business applications, databases and object oriented software engineering.

Roxana-Marina STRAINU graduated in 2014 the Master of Business Information Systems at the Faculty of Economics and Business Administration, Alexandru Ioan Cuza University of Iasi. She also graduated the Faculty of Mathematics in the year 2005. She is interested in developing smart systems and mobile applications on Android platform. Now she is a PhD student in the business information systems area.

Alexandra NISTOR graduated the Faculty of Economics and Business Administration in 2011 and the Master of Business Information Systems at the Faculty of Economics and Business Administration in 2013. Her research interests include the use of automated testing in small and medium companies. Now she is a PhD student in the business information systems area.

DOI: 10.12948/issn14531305/22.1.2018.04