Semantic Web Services for University Course Registration

Şengül Çobanoğlu* and Zeki Bayram**

Computer Engineering Department,
Eastern Mediterranean University,
Famagusta,
North Cyprus
{sengul.cobanoglu,zeki.bayram}@emu.edu.tr
http://cmpe.emu.edu.tr/

Abstract. Semantic web services, with proper security procedures, have the potential to open up the computing infrastructure of an organization for smooth integration with external applications in a very controlled way, and at a very fine level of granularity. Furthermore, the process of using the provided functionality, consisting of discovery, invocation and execution of web services may be automated to a large extent. In this paper, we show how semantic web services and service-oriented computing can facilitate this integration in the education domain. Specifically, we use the Rule variant of Web Services Modeling Language (WSML) to semantically specify the functionality of web services for on-line course registration, a goal for consuming the provided functionality, as well as the ontologies needed for a shared terminology between the service and goal.

Keywords: Semantic web services, ontology, course registration, Web Service Modeling Language, Web Service Modeling Ontology, discovery, service orientation

1 Introduction

Service-oriented architecture [1], consisting of loosely coupled modular components with standard interfaces, can facilitate greatly the process of developing large and complex software in a distributed manner. In this paradigm, due to the standard interfaces of the components, the provided functionality is platform independent. Furthermore, new applications can easily be "assembled" from existing components by providing a user interface and some logic combining the functionalities of the components used and synthesizing the required functionality. The problem with this seemingly ideal approach to software development,

* Ş. Çobanoğlu is a graduate of the Department of Computer Engineering, Eastern Mediterranean University, Famagusta, Cyprus.

** Z. Bayram is with the Department of Computer Engineering, Eastern Mediterranean University, Famagusta, Cyprus.

which naturally promotes software re-usability and platform independence, is that as the number of modules get larger and larger, finding a web service with the desired functionality becomes harder and harder. This is where semantic web services enter the picture and come to help. Web service functionality can be described semantically using special purpose web service description languages, and desired functionality can also be specified semantically in the form of goals. A matcher can then be used for matching the desired functionality as specified in the goal, with the functionalities provided by various web services. Once a satisfactory match is found, invocation can be made, either in automatic mode, or manually, or somewhere in between.

In this paper, we show the potential beneficial use of semantic web services [2] in the educational domain, more specifically for making course registrations. In previous work, semantic web service applications were investigated in the banking domain [3][4], as well as the health domain [5].

Before we go into the specifics of our application, we present below terminology and concepts relevant to semantic web services. The notion of an ontology is used widely in the semantic web. Ontologies are used for representing shared domain knowledge [6][7]. They form an important foundation of the semantic web on which other components are built [8]. The main component of an ontology is the "concept," which is similar in essence to classes in object-oriented programming languages [9]. Web services are computational units on the World Wide Web (WWW), and can be called through standard interfaces and protocols, such as HTTP [10] and SOAP [11]. Web services also allow an organization to open up part of its internal computing infrastructure to the outside world in a controlled way. A drawback of "normal" web services is their purely syntactic specification, which makes reliable automatic discovery impossible. *Semantic* Web services attempt to remedy the purely syntactic specification of regular web services by providing rich, machine interpretable semantic descriptions of Web services [12] so that the whole process of web service discovery, orchestration or composition, and execution can be automated through appropriate semantic web service frameworks.

Web Service Modeling Ontology (WSMO) [13] is a framework for semantic description of Semantic Web Services based on the Web Service Modeling Framework (WSMF) [14]. Its four main components are ontologies, web services, goals and mediators. Web Service Modeling Language (WSML) [15] is a language for modelling web services, ontologies, and related aspects of WSMO framework, to provide the description of semantic web services so that automatic discovery and invocation becomes possible. Five language variants of WSML exist, which are WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full.

In this paper, we use WSML-Rule to semantically specify an online university course registration web service. We first define an ontology which contains relevant concepts, axioms, relations and instances. Then we proceed to semantically define the web service itself, and then a goal to consume the web service.

The remainder of the paper is organized as follows. Section 2 describes the semantic specification of a web service for course registration in WSML-Rule. This

specification includes an ontology, consisting of concepts, instances, relations and axioms of the course registration domain, the functional specification of the web service itself in terms of preconditions, postconditions, assumptions and effects, as well as a goal for consuming the web service. Section 3 contains a discussion of the problems encountered and shortcomings of the WSML language. Finally, section 4 is the conclusion and future work.

2 Specifying course registration functionality in WSML

The web service whose functionality we want to semantically describe implements a course registration use-case, where a student tries to register for a course in a given semester. Several conditions must be satisfied before the registration can take place. In the following sub-sections, we present the ontology, goal and web service specification for the course registration use-case. Note that we used WSML-Rule throughout the specification, since it supports rule-based logic programming, with which we can write axioms that act as integrity constraints, and also do computations.

2.1 Course Registration Ontology

The course registration ontology consists of *concepts*, *relations*, *instances*, *relationInstances* and *axioms*. We have placed concepts and instances into different files and used the importing facility of WSML to better manage the ontology. Figure 1 depicts the prologue of the WSML file containing the ontological structures used in the specification. Note the choice of WSML variant selected (WSML-Rule), the definition of the namespace property, the beginning of the registration ontology, and the importation of two other ontologies containing instances.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://cmpe.emu.edu.tr/courseRegistration#",
     discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#",
     dc       _"http://purl.org/dc/elements/1.1#" }

ontology courseRegistration
     importsOntology {courseRegistrationInstances, courseRegistrationRelationInstances}
```

Fig. 1. Prologue of the course registration ontology file

Concepts in the registration ontology Figure 2 graphically depicts the concepts of the ontology. In figure 3 we have the concept definitions for University,

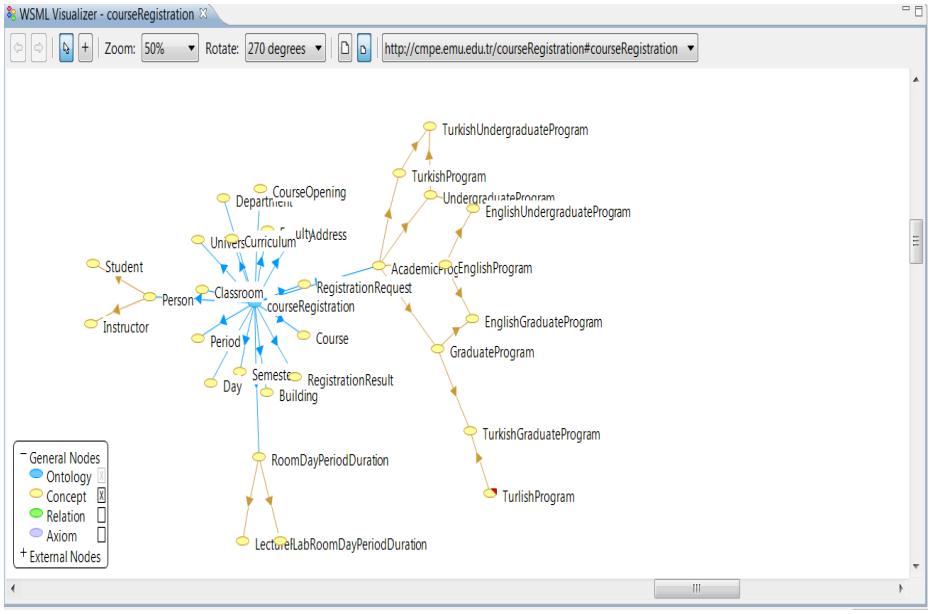

Fig. 2. Concepts of CourseRegistration ontology (graphical representation)

```
concept University
     uname ofType (1 *) _string
     locatedAt ofType  (1 *) Address

concept Address
     street ofType  (0 1) _string
     city ofType  (0 1) _string

concept Faculty
     facultyName ofType  (0 1) _string
     atUniversity ofType  (0 1) University

concept Department
     deptID ofType  (0 1) _string
     deptName ofType  (0 1) _string
     inFaculty ofType  (0 1) Faculty

concept AcademicProgram
     programName ofType  (0 1) _string
     programID ofType  (0 1) _string
     belongsTo ofType  (0 1) Department

concept Curriculum
     academicProgram ofType  (0 1) AcademicProgram
     refCode ofType  (0 1) _string
     courseName ofType  (0 1) Course
```

Fig. 3. Registration ontology concepts (part 1)

```
concept UndergraduateProgram subConceptOf AcademicProgram
concept GraduateProgram subConceptOf AcademicProgram
concept TurkishProgram subConceptOf AcademicProgram
concept EnglishProgram subConceptOf AcademicProgram
concept EnglishGraduateProgram subConceptOf {EnglishProgram, GraduateProgram}
concept EnglishUndergraduateProgram subConceptOf { EnglishProgram, UndergraduateProgram}
concept TurkishGraduateProgram subConceptOf {TurlishProgram, GraduateProgram}
concept TurkishUndergraduateProgram subConceptOf { TurkishProgram, UndergraduateProgram}
```

Fig. 4. Some sub-concept relationships

Address, Faculty, Department, and AcademicProgram and Curriculum. A department may be running more than one program, and for each program, we have a curriculum, with a list of courses.

Figure 4 shows the sub-concept relationships among concepts. For example, EnglishProgram is a sub-concept of AcademicProgram. Note how multiple inheritance is possible.

Figure 5 depicts concepts needed for course registration specifically. Note the difference between Course and CourseOpening concepts. The former gives information about a course in general, and the latter is used when a course is offered in a given semester. Semester, Day, and Period are *utility* concepts.

Figure 6 depicts the definition of the Person concept, and its two sub-concepts Student and Instructor. The *yearEnrolled* attribute of Student is the year that the student enrolled university and the *enrolledIn* attribute is the Academic program in which student is enrolled. Of special interest is the *tookCourse* attribute, since it will be indirectly defined through the *TookRelation* axiom, given in figure 12.

Finally in figure 7, we have two concepts, RegistrationRequest and RegistrationResult that are used to pass information to the the web service, and get a confirmation back. The request is for a course in a given year and semester. The confirmation is the registration into a specific course opening.

Relations Although attributes of a concept act like relationships between the host object and some other object, *n-ary* relationships where $n > 2$ are best represented using explicit relations, which WSML provides as a construct. Sometimes we use binary relationships for convenience, and it is also possible to establish a connection between attributes of a concept and relations explicitly through axioms.

In figure 8, we see the four relations present in the ontology. These are *teaches* for specifying which instructor teaches which course opening, *takes* for students for specifying which student takes which course opening, *tookCourse* for recording which student has already taken which course, and *prerequisite* for recording the prerequisite relationship among courses.

```
concept Course
     courseCode ofType  (0 1) _string
     courseName ofType _string
     hasPrerequisite ofType (0 *) Course
     lecture_hour ofType  (0 1) _integer
     tutorial_hour ofType  (0 1) _integer
     credits ofType  (0 1) _integer
     ects ofType  (0 1) _integer
     belongsToProgram ofType  (0 1) UndergraduateProgram

concept CourseOpening
     groupNo ofType  (0 1) _integer
     ofCourse ofType  (0 1) Course
     semester ofType  (0 1) Semester
     id_courseOpening ofType  (0 1) _string
     teaching_times ofType  (1 4) RoomDayPeriodDuration
     current_size ofType (0 1) _integer
     year ofType  (0 1) _integer

concept Classroom
     classID ofType  (0 1) _string
     location ofType  (0 1) Building
     capacity ofType (0 1) _integer
     inDepartment ofType  (0 1) Department
     roomNumber ofType  (0 1) _string

concept Semester
concept Day
concept Period
concept Building

concept RoomDayPeriodDuration
     room ofType  (1 1) Classroom
     day ofType  (1 1) Day
     period ofType  (1 1) Period
     duration ofType  (1 1) _integer


concept LectureRoomDayPeriodDuration subConceptOf RoomDayPeriodDuration
concept LabRoomDayPeriodDuration subConceptOf RoomDayPeriodDuration
```

Fig. 5. Registration ontology concepts (part 2)

```
concept Person
     ID ofType  (0 1) _string
     gender ofType  (0 1) _string
     Date_of_Birth ofType  (0 1) _date
     name ofType  (0 1) _string
     lastName ofType  (0 1) _string
     address ofType Address

concept Student subConceptOf Person
     yearEnrolled ofType  (0 1) _integer
     overallCGPA ofType  (0 1) _float
     enrolledIn ofType  (0 2) AcademicProgram
     semesterEnrolled ofType  (0 1) Semester
     tookCourse_ ofType (0 *) Course
           nfp
                 dc#relation hasValue {TookRelation}
            endnfp

concept Instructor subConceptOf Person
     works_in ofType  (1 *) Department
```

Fig. 6. Registration ontology concepts (part 3)

```
concept RegistrationRequest
     student ofType Student
     course ofType Course
     year ofType _integer
semester ofType Semester

concept RegistrationResult
     student ofType Student
     courseOpening ofType CourseOpening
```

Fig. 7. Registration ontology concepts (part 4)

```
relation teaches( ofType Instructor,  ofType CourseOpening)
relation takes( ofType Student,  ofType CourseOpening)
relation tookCourse(ofType Student, ofType Course)
relation prerequisite(ofType Course,ofType Course)
```

Fig. 8. Relations in the course registration ontology

Concept and relation instances Instances are actual data in the ontology. In figure 9 we give a representative set of instances for several concepts to enable better understanding of the way in which concepts and relations are used in the ontology. Figure 10 has a representative set of relation instances.

Axioms An axiom in WSML is a logical statement which can be used to implicitly define concepts and conditions for concept membership (i.e. instances), specify relations on ontology elements as well as assert any kind of constraints regarding the ontology. Axioms that specify forbidden states of the ontology start with "!-", and they function like integrity constraints of databases.

Figure 11 depicts four named axioms. The "registrationRules" axiom defines the "clashes", "prerequisiteNotTaken" and "classSizeExceeded" predicates. The "clashes" predicate is true when two course openings in a given semester are taught in the same physical location at exactly the same time. The "prerequisiteNotTaken" predicate is true when a student is taking a course without having taken its prerequisite course. Note the use of the *naf* operator in WSML, which performs "negation as failure", the operational counterpart of logical negation. The "classSizeExceeded" predicate is true if the number of students registered to a course opening has exceeded the capacity of the room in which the course in taught. The "noClashRoom" axiom is a constraint, forbidding any clashes of rooms, which means the same room cannot be assigned to two courses at exactly the same time slot. Similarly, "prerequisiteTaken" forbids taking a course without first taking is prerequisite course, and "classSizeViolation" forbids exceeding class sizes when registering students.

Figure 12 depicts the remaining axioms of the ontology. The "noClashTeacher" axiom makes sure that the meeting times of two courses taught by a teacher have no overlap, since naturally a person cannot be present in two places at the same time. Axiom "noClashStudent" imposes a similar constraint for students, although this restriction may be somewhat unrealistic in a university environment. Finally, axiom "TookRelation" establishes a connection between the "tookCourse" relation and the tookCourse_ attribute of Student objects: if a student s and a course c are in the tookCourse relation, then the tookCourse_ attribute value of s is c.

2.2 Goals

A goal is a declarative statement of what the requester can provide, and what it expects as a result of invoking a web service. The "what I can provide" part is specified in the "precondition" of the goal (this is an unfortunate terminology, since it implies that the goal has a precondition), and the "what I expect" part is specified in the "postcondition". Goals are like elaborate queries in a database system, except they can be far more complex, since they can involve any logical statement, both in the "precondition" part and the "postcondition" part. A goal is "matched" against web service specifications, and the one that "best" satisfies the requirements of the goal is chosen for execution. The definition of "best" can

```
instance dept_computer_engineering memberOf Department
     deptName hasValue "Computer Engineering"
     inFaculty hasValue faculty_of_engineering
     deptID hasValue "computer_engineering"

instance cmpe_undergrad_eng memberOf EnglishUndergraduateProgram
     programID hasValue "cmpe_undergrad_eng"
     programName hasValue "Computer Engineering Undergraduate English"
     belongsTo hasValue dept_computer_engineering

instance curriculum_cmpe_undergrad_eng memberOf Curriculum
     academicProgram hasValue cmpe_undergrad_eng
     refCode   hasValue  "cmpecurriculum"
     courseName hasValue cmpe318

instance cmpe354 memberOf Course
     belongsToProgram hasValue cmpe_undergrad_eng
     courseName hasValue "Introduction to Databases"
     courseCode hasValue "cmpe354"
     hasPrerequisite hasValue cmpe211
     ....

instance cmpe354_spring_2012_gr1 memberOf CourseOpening
     id_courseOpening hasValue "cmpe354_spring_2012"
     year hasValue 2012
     semester hasValue spring
     ofCourse hasValue cmpe354
     groupNo hasValue 1
     current_size hasValue 4
     teaching_times hasValue {lecture_cmpe128_wednesday_per2_2,
                              lab_cmpelab5_friday_per4_2 }

instance room_cmpe128 memberOf Classroom
     inDepartment hasValue dept_computer_engineering
     classID hasValue "room_cmpe128"
     capacity hasValue 60
     location hasValue cmpe_building
     ....

instance lecture_cmpe128_monday_per2_2 memberOf LectureRoomDayPeriodDuration
     room hasValue room_cmpe128
     day hasValue monday
     period hasValue per2
     duration hasValue 2

instance ayse memberOf Student
     ID hasValue "104059"
     name  hasValue  "Ayse"
     lastName  hasValue  "Akcam"
     ....

instance spring memberOf Semester
instance monday memberOf Day
instance per1 memberOf Period
instance cmpe_building memberOf Building
```

Fig. 9. Various concept instances in the course registration ontology

```
relationInstance  teaches(ali, cmpe354_spring_2012_gr1)
relationInstance  takes(ayse, cmpe354_spring_2012_gr1)
relationInstance prerequisite(cmpe211,cmpe354)
relationInstance tookCourse(zainab,cmpe211)
```

Fig. 10. Various relation instances in the course registration ontology

```
axiom registrationRules
   definedBy
      clashes(?co1,?co2):-
         ?co1 memberOf CourseOpening
         and ?co2 memberOf CourseOpening
         and ?co1 != ?co2
         and ?co1[teaching_times hasValue ?tt1, year hasValue ?y1, semester hasValue ?s1]
         and ?co2[teaching_times hasValue ?tt1, year hasValue ?y1, semester hasValue ?s1].

      prerequisiteNotTaken(?student,?course,?precourse):-
                                 takes(?student, ?courseOpening) and
                                 ?courseOpening[ofCourse hasValue ?course] memberOf CourseOpening and
                                 prerequisite(?pre,?course) and
                                 naf tookCourse(?student,?precourse).

     classSizeExceeded:- ?co [teaching_times hasValue ?tt,current_size hasValue ?s] memberOf CourseOpening and
                           ?tt[room hasValue ?room] memberOf RoomDayPeriodDuration and
                           ?room[capacity hasValue ?maxCap] and
                           ?s > ?maxCap.

axiom noClashRoom
   definedBy
          !-clashes(?x,?y).

axiom prerequisiteTaken
   definedBy
          !- prerequisiteNotTaken(?student,?course,?pre).

axiom classSizeViolation
   definedBy
          !- classSizeExceeded.
```

Fig. 11. Some axioms of the registration ontology

```
axiom noClashTeacher
   definedBy
      !- ?t1 memberOf Instructor
         and ?co1 memberOf CourseOpening
         and ?co2 memberOf CourseOpening
         and teaches(?t,?co1)
         and teaches(?t,?co2)
         and ?co1 != ?co2
         and ?co1[teaching_times hasValue ?tt1, year hasValue ?y1, semester hasValue ?s1]
         and ?co2[teaching_times hasValue ?tt2, year hasValue ?y1, semester hasValue ?s1]
         and ?tt1[day hasValue ?d1, period hasValue ?p1]
         and ?tt2[day hasValue ?d1, period hasValue ?p1].

axiom noClashStudent
   definedBy
      !- ?t1 memberOf Student
         and ?co1 memberOf CourseOpening
         and ?co2 memberOf CourseOpening
         and takes(?t1,?co1)
         and takes(?t1,?co2)
         and ?co1[teaching_times hasValue ?tt1, year hasValue ?y1, semester hasValue ?s1]
         and ?co2[teaching_times hasValue ?tt2, year hasValue ?y1, semester hasValue ?s1]
         and ?tt1[day hasValue ?d1, period hasValue ?p1]
         and ?tt2[day hasValue ?d1, period hasValue ?p1]
         and ?co1 != ?co2.

axiom TookRelation
   definedBy
      ?x[tookCourse_ hasValue ?course] memberOf Student:- tookCourse(?x,?course).
```

Fig. 12. More axioms of the course registration ontology

change, and non-functional requirements can also play a role in which web service is chosen. Execution can be automatic if an appropriate semantic web service framework is used. Ontologies are used to establish a "common understanding" of terminology between goals and web services.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://cmpe.emu.edu.tr/courseRegistration#",
            discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#",
            dc _"http://purl.org/dc/elements/1.1#" }

goal goalCourseRegistration
   importsOntology
      { _"http://cmpe.emu.edu.tr/courseRegistration#courseRegistration",
        _"http://cmpe.emu.edu.tr/courseRegistration#courseRegistrationInstances",
        _"http://cmpe.emu.edu.tr/courseRegistration#courseRegistrationRelationInstances"}

   capability goalCourseRegistrationAliCapability
      nonFunctionalProperties
         discovery#discoveryStrategy hasValue {discovery#HeavyWeightRuleDiscovery,
                                               discovery#NoPreFilter}
      endNonFunctionalProperties

      sharedVariables {}

   precondition
      definedBy
         ?rr[student hasValue ali,
             course hasValue cmpe354,
             year hasValue 2012,
             semester hasValue spring] memberOf RegistrationRequest.

   postcondition
      definedBy
         ?aResult[student hasValue ali,courseOpening hasValue ?co]
                                               memberOf RegistrationResult
               and ?co [ofCourse hasValue cmpe354,
                                year hasValue 2012,
                                semester hasValue spring,
                  groupNo hasValue ?groupno ] memberOf CourseOpening.
```

Fig. 13. Goal for registering a specific student to a specific course in a given year and semester

In figure 13, we have a goal for registering a specific student, Ali, to the CMPE354 course in the spring 2012 semester. The goal specification starts with stating the variant of WSML that will be used (WSML-rule), namespaces, and imported ontologies which contain concept definitions and instances. In the capability section, non-functional properties state that "heavyweight rule matching" should be used in the discovery process, meaning that full logical inference should be the method utilized. The shared variables section should contain logical variables that are shared in all parts of the goal (e.g. the precondition and postcondition). In this case, since there are no shared variables, this section is

empty. The "precondition" of the goal contains the information needed to register the student, i.e. an instance of the "RegistrationRequest" concept. This instance acts like the parameter to the web service call. The postcondition states that an instance of the "RegistrationResult" concept is required, which acknowledges that the requested registration has been done. The variables *?co* and *?aResult* are logic variables that are meant to be bound to values as a result of a successful invocation of a web service.

2.3 Web Services

The web service specification in WSML describes in logical terms what the web service expects form the requester, the state of the world which must exist before it can be called, as well as results it can return to the requester, and the changes it can cause to the state of the world after it has finished its execution. *Preconditions* specify what information a web service requires from the requester. *Assumptions* describe the state of the world which is assumed before the execution of the Web service. *Postconditions* are statements about the existence of objects in the ontology created by the web service to be returned to he requester. *Effects* describe the state of the world that is guaranteed to be reached after the successful execution of the Web service.

In figure 14, we have the course registration web service specification. It starts out, similarly to the goal specification, with the WSML variant used and the namespaces. Three ontologies are imported as in the goal. There are five shared variables in the capability. They are implicitly universally quantified. These variables are *?student*, *?course*, *?year*, *?semester*, *?oldsize* and *?co*. The precondition instantiates its variables with the data provided by the requester, and through shared variables passes them to the other parts of the specification. For example, the *?course* variable's contents are used in the assumption to find a CourseOpening instance for that course. In the assumption, the current size of the CourseOpening instance is stored in the *?oldsize* variable, which is used in the effect to increase the current size by 1. Note that the effect is not "performing" an action, but merely declaring the state of the world after the web service has finished execution, which is that the current size in the course opening is increased by 1, and that now the student is considered to be registered to the course opening. Lastly, the postcondition states the existence of an instance of the RegistrationResult concept in the ontology that will be "sent" to requester.

3 Discussion

Although WSML-Rule is a relatively comprehensive language, we have still found several constructs that are missing from it, the presence of which would have made the language much more powerful and expressive. Some of these are:

- *Lack of aggregate operators, such as "sum", "count" etc.* Due to this missing feature, we could not check in a declarative way that the capacity of each classroom should not be exceeded through registrations.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"

namespace { _"http://cmpe.emu.edu.tr/courseRegistration#",
     discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#",
     dc       _"http://purl.org/dc/elements/1.1#" }

webService web_service_courseRegistration

   importsOntology {courseRegistration, courseRegistrationInstances,
                    courseRegistrationRelationInstances}

   capability web_service_courseRegistrationCapability

      nonFunctionalProperties
         discovery#discoveryStrategy hasValue discovery#HeavyweightDiscovery
      endNonFunctionalProperties

      sharedVariables {?student, ?course,?year, ?semester, ?oldsize, ?co}

      assumption
         definedBy
            ?co[ofCourse hasValue ?course,
                    year hasValue ?year,
                     semester hasValue ?semester,
                     groupNo hasValue ?groupno,
                     current_size hasValue ?oldsize] memberOf CourseOpening.

      precondition
         definedBy
            ?rr[student hasValue ?student,
                course hasValue ?course,
                year hasValue ?year,
                semester hasValue ?semester] memberOf RegistrationRequest.
      effect
         definedBy
            takes(?student,?co) and
            ?co[current_size hasValue (?oldsize+1)].

      postcondition
         definedBy
            ?aResult[student hasValue ?student,
                     courseOpening hasValue ?co] memberOf RegistrationResult.
```

Fig. 14. Specification of the course registration web service

- *Lack of an exception mechanism.* Currently there is no way to deal with constraint violations in the ontology. They just get reported as errors.
- *Inability to specify different postconditions and effects depending on the preconditions and assumptions.* Currently, a web service specification is like an "if-then" statement of programming languages: if the preconditions and assumptions are correct, then the web service guarantees the effect and postcondition. What would be much more useful is a "case" structure, where more than one set of precondition-assumption pairs would exist, with their corresponding postcondition-effect pairs. The ability to have more than one capability for a web service would solve this problem.
- *Inability to have more than one outcome for one set of conditions in which the web service is called.* It would be desirable to be able to associate more than one postcondition-effect pair for the *same* precondition-assumption pair. We need to be able to say that the result of calling the web service can be $< postcondition_1, effect_1 >$ or $< postcondition_2, effect_2 >$ or $\ldots$ or $< postcondition_n, effect_n >$, but this is not possible. For example, currently we cannot deal with the case that a request to register to a course may fail, in which case the state of the world will not change as a result of the web service call.

4 Conclusion and Future Research Directions

We have specified semantically, using WSML-Rule, a web service for course registration. This involved the definition of an ontology, with concepts needed to capture domain knowledge and axioms to implement integrity constraints, as well the specification of web service capability and goal to consume the provided service. Our work has revealed several weaknesses of WSML-Rule, which we identified above.

For future work, we are planning to work on remedying the weak points of WSML-Rule. We believe semantic web services, more specifically WSMO and WSML, when properly fixed and updated to answer the discovered deficiencies, can be the backbone of the future semantic web with intelligent agents.

References

1. Erl, T.: Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall (2005)
2. Dieter Fensel, D., Facca, F.D., Simperl, E., Toma, I.: Semantic Web Services. Springer (2011)
3. Sharifi, 0., Bayram, Z.: Database Modelling Using WSML in the Specification of a Banking Application. In: Proceedings of WASET 2013, pp. 263–267, Zurich, Switzerland (2001)
4. Sharifi, 0., Bayram, Z.: Specifying Banking Transactions using Web Services Modeling Language (WSML). In: The fourth International Conference on Information and Communication Systems (ICICS 2013), pp. 138–143, Irbid,Jordan (2001)
5. Sharifi, 0., Bayram, Z.: A Critical Evaluation of WSMO and WSML through an E-health Semantic Web Service Specification Case Study. Submitted for publication (2013)
6. Domingue, J., Cabral, L,. Galizia, S., Tanasescu, V., Gugliotta, A., Norton, B., Pedrinaci, C.: IRS-III: A broker-based approach to semantic Web services. Web Semantics: Science, Services and Agents on the World Wide Web, vol. 6, num. 2, pp. 109–132 (2008).
7. Gruber, T. R.: A translation approach to portable ontology specifications. Knowledge Acquisition, vol. 5, num. 2, 199–220 (1993)
8. Berners-Lee, T., Hendler, J.: Scientific publishing on the semantic web. Nature, vol. 410, pp. 1023–1024 (2001)
9. Lara, R., Roman, D., Polleres, A., Fensel, D.: A conceptual comparison of WSMO and OWL-S. In: Zhang, L.J., Jeckle, M.. (eds.) ECOWS 2004. LNCS, vol. 3250, pp. 254–269. Springer, Heilderberg (2004)
10. Hypertext Transfer Protocol Overview, `http://www.w3.org/Protocols`
11. SOAP specifications, `http://www.w3.org/TR/soap`
12. Fensel, D., Lausen, H., Polleres, A., Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: The Web Service Modeling Ontology . Springer-Verlag, New York (2006)
13. Web Service Modeling Ontology(WSMO), `http://www.w3.org/Submission/WSMO`
14. Fensel, D., Bussler, C.: The web service modeling framework WSMF. Electronic Commerce Research and Applications, vol. 1, num. 2, pp. 113–137 (2002)
15. D16.1v1.0 WSML Language Reference, `http://www.wsmo.org/TR/d16/d16.1`