A Service-Oriented Approach for Curriculum Planning and Validation

Matteo Baldoni[1], Cristina Baroglio[1], Ingo Brunkhorst[2], Elisa Marengo[1], Viviana Patti[1]

[1] Dipartimento di Informatica — Università degli Studi di Torino
c.so Svizzera, 185, I-10149 Torino (Italy)
{baldoni,baroglio,patti}@di.unito.it, elisa.mrng@gmail.com
[2] L3S Research Center, University of Hannover
D-30539 Hannover, Germany
brunkhorst@l3s.de

Abstract. We present a service-oriented personalization system, set in an educational framework, based on a semantic annotation of courses, given at a knowledge level (what the course teaches, what is requested to know for attending it in a profitable way). The system supports users in building personalized curricula, formalized by means of an action theory. It is also possible to verify the compliance of curricula w.r.t. a model, expressing constraints at a knowledge level. For what concerns the first task, classical planning techniques are adopted, which take into account both the student's initial knowledge and her learning goal. Instead, curricula validation is done against a model, formalized as a set of temporal constraints. We have developed a prototype of the planning and validation services, by using -as reasoning engines- SWI-Prolog and the SPIN model checker. Such services will be supplied and combined as plug-and-play personalization services in the Personal Reader framework.

1 Introduction and Motivation

The birth of the Semantic Web brought along standard models, languages, and tools for representing and dealing with machine-interpretable semantic descriptions of Web resources, by giving a strong new impulse to research on personalization. The introduction of machine-processable semantics makes the use of a variety of reasoning techniques for implementing personalization functionalities possible, widening the range of the forms that personalization can assume. So far, reasoning in the Semantic Web is mostly reasoning about knowledge expressed in some ontology. However personalization may involve also other kinds of reasoning and knowledge representation, that conceptually lie at the logic and proof layers of the Semantic Web tower.

Moreover, the next Web generation promises to deliver Semantic Web Services, that can be retrieved and *combined* in a way that satisfies the user. It opens the way to many forms of *service-oriented personalization*. Web services provide an ideal infrastructure for enabling *interoperability* among personalization applications and for constructing Plug&Play-like environments, where the user can

select and combine the kinds of services he or she prefers. Personalization can be obtained by taking different approaches, e.g. by developing services that offer personalization functionalities as well as by personalizing the way in which services are selected, and *composed* in order to meet specific user's requirements.

In the last years we carried on a research in the educational domain, by focussing on *semantic web* representations of learning resources and on *automated reasoning* techniques for enabling different and complementary personalization functionalities, e.g. curriculum sequencing [6, 7] and verification of the compliance of a curriculum against some course design goals [5]. Our current aim is to implement such results in an organic system, where different personalization services, that exploit semantic web reasoning, can be combined to support the user in the task of building a curriculum, based on *learning resources* that represent courses.

While in early times learning resources were simply considered as "contents", strictly tied to the platform used for accessing them, recently, greater and greater attention has been posed on the issue of *re-use* and of a *cross-platform* use of educational contents. The proposed solution is to adopt a *semantic annotation* of contents based on standard languages, e.g. RDF and LOM. Hereafter, we will consider a *learning resource* as formed by *educational contents* plus *semantic meta-data*, which supply information on the resources at a *knowledge level*, i.e. on the basis of concepts taken from an ontology that describes the educational domain. In particular we rely on the interpretation of learning resources as *actions* discussed in [6, 7]: the meta-data captures the *learning objectives* of the learning resource and its *pre-requisites*. By doing so,one can rely on a classical theory of actions and apply different reasoning methods -like *planning*- for building personalized curricula [6, 7]. The modeling of learning resources as actions also enables the use of model checking techniques for developing a validation service that detects if a user-given curriculum is compliant w.r.t an abstract model, given as a set of constraints. In the following we present our achievements in the implementation of a Planning service and a Validation service that can interoperate within the Personal Reader Framework [18].

Curriculum planning and validation offer a useful support in many practical contexts and can be fruitfully combined for helping students or teaching institutions. Often a student knows what competency he/she would like to acquire but has no knowledge of which courses will help him/her acquiring it. Moreover, taking courses at different Universities is becoming more and more common in Europe. As a consequence, building a curriculum might become a complicated task for students, who must deal with an enormous set of courses across the European countries, each described in different languages and on the basis of different keywords.

The need of personalizing the sequencing of learning resource, w.r.t. the student's interests and context, has often to be *combined* with the ability to check that the resulting curriculum *complies* against some abstract *curricula specification*, which encodes the *curricula-design goals* expressed by the teachers or by the institution offering the courses. Consider a student, who wants to build

a valid curriculum with the support of our automatic system. The student can either use as a basis the suggestion returned by the system or he/she can design the curriculum by hand, based on own criteria. In both cases a personalized curriculum is obtained and can be given in input to the validation service for checking the compliance against a curricula model. Curricula models specify general rules for building learning paths and can be interpreted as constraints designed by the University for guaranteeing the achievement of certain learning goals. These constraints are to be expressed in terms of knowledge elements, and maybe also on features that characterize the resources.

Consider now a university which needs to certify that the specific curricula, that it offers for achieving a certain educational goal, and that are built upon the courses offered locally by the university itself, respect some European guidelines. In this case, we could, in fact, define the guidelines as a set of constraints at an abstract level, i.e. as relations among a set of competencies which should be offered in a way that meets some given scheme. At this point the verification could be performed automatically, by means of a proper reasoner. Finally, the automatic checking of compliance combined with curriculum planning could be used for implementing processes like cooperation among institutes in curricula design and integration, which are actually the focus of the so called *Bologna Process* [15], promoted by the EU.

While SCORM [2] and Learning Design [19, 20] represent the most important steps in the direction of managing and using e-learning based courses and workflows among a group of actors participating in learning activities, most of the available tools lack the machine-interpretable information about the learning resources, and as a result they are not yet open for reasoning-based personalization and automatic composition and verification. Given our requirements, it is a natural choice to settle our implementation in the Personal Reader (PR) framework. The PR relies on a service-oriented architecture enabling personalization, via the use of semantic *Personalization Services*. Each service offers a different personalization functionality, e.g. recommendations tailored to the needs of specific users, pointers to related (or interesting or more detailed/general) information, and so on. These semantic web services communicate solely based on RDF documents.

The paper is organized as follows. Section 2 describes our approach to the representation and reasoning about learning resources, curricula, and curricula models. The implementation of the two services and their integration into the PR Framework is discussed in section 3. We finish with conclusions and hints on future work in Section 4.

2 Curricula representation and reasoning

Let us begin with the introduction of our approach to the representation of learning resources, curricula, and curricula models. The basic idea is to describe all the different kinds of objects, that we need to tackle and that we will introduce hereafter, on the basis of a set of predefined *competencies*, i.e. terms identifying

specific *knowledge elements*. We will use the two terms as synonyms. Competencies can be thought of, and implemented, as concepts in a shared ontology. In particular, for what concerns the application system described here, competencies were extracted by means of a semi-automatic process and stored as an RDF file (see Section 3.1 for details).

Given a predefined set of competencies, the initial knowledge of a student can be represented as a set of such concepts. This set changes, typically it grows, as the student studies and learns. In the same way, a user, who accesses a repository of learning resources, does it with the aim of finding materials that will allow him/her to acquire some knowledge of interest. Also this knowledge, that we identify by the term *learning goal*, can be represented as a set of knowledge elements. The learning goal is to be taken into account in a variety of tasks. For instance, the construction of a personalized curriculum is, actually, the construction of a curriculum which allows the achievement of a learning goal expressed by the user. In Section 3 we will describe a *curricula planning service* for accomplishing this task.

2.1 Learning resources and curricula

A *curriculum* is a sequence of *learning resources* that are homogeneous in their representation. Based on work in [6, 7], we rely on an *action theory*, and take the abstraction of resources as *simple actions*. More specifically, a learning resource is modelled as an action for acquiring some competencies (called *effects*). In order to understand the contents supplied by a learning resource, the user is sometimes required to own other competencies, that we call *preconditions*. Both preconditions and effects can be expressed by means of a *semantic annotation* of the learning resource [7]. In the following we will often refer to learning resources as "courses" due to the particular application domain that we have considered (university curricula).

As a simple example of "learning resource as action", let us, then, report the possible representation (in a classical STRIPS-like notation) of the course "databases for biotechnologies" (*db_for_biotech* for short):

ACTION: db_for_biothec(),
PREREQ: relational_db, EFFECTS: scientific_db

The prequisites to this action is to have knowledge about *relational databases*. Its effect is to supply knowledge about *scientific databases*.

Given the above interpretation of learning resources, a *curriculum* can be interpreted as a *plan*, i.e. as a sequence of actions, whose execution causes transitions from a state to another, until some final state is reached. The *initial state* contains all the competences that we suppose available before the curriculum is taken, e.g. the knowledge that the student already has. This set can also be empty. The *final state* is sometimes required to contain specific knowledge elements, for instance, all those that compose the user's learning goal. Indeed, often curricula are designed so to allow the achievement of a well-defined *learning goal*.

A transition between two states is due to the application of the action corresponding to a learning resource. Of course, for an action to be applicable, its preconditions must hold in the state to which it should be applied. The application of the action consists in an *update* of the state. We assume that competences can only be added to states. Formally, we assume that the domain is monotonic. The intuition behind this assumption is that the act of using a new resource will never erase from the students' memory the concepts acquired insofar. Knowledge grows incrementally.

2.2 Curricula models

Curricula models consist in sets of constraints that specify desired properties of curricula. Curricula models are to be defined on the basis of knowledge elements as well as of learning resources (courses). In particular, we would like to restrict the set of possible sequences of resources corresponding to curricula. This will be done by imposing constraints on the *order* by which knowledge elements are added to the states (e.g. "a knowledge element α is to be acquired before a knowledge element β"), or by specifying some *educational objectives* to be achieved, in terms of knowledge that must be contained in the final state (e.g. "a knowledge element α must be acquired sooner or later"). Therefore, we represent a curricula model as a set of *temporal constraints*. Being defined on knowledge elements, a curricula model is *independent* from the specific resources that are taken into account, for this reason, it can be *reused* in different contexts and it is suitable to open and dynamic environments like the web.

The possibility of *verifying the compliance of curricula to models* is extremely important in many applicative contexts, as explained by examples in the introduction. In some cases these checks could be integrated into the curriculum construction process; nevertheless, it is important to be able to perform the verification independently from the construction process. Let us consider again our simple scenario concerning a university, which offers a set of curricula that are proved to satisfy the guidelines given by the EU for a certain year. After a few years, the EU guidelines change: our University has the need to check if the curricula that it offers, still satisfy the guidelines, without rebuilding them.

A natural choice for representing temporal constraints on action paths is linear-time temporal logic (LTL) [14]. This kind of logic allows to verify if a property of interest is true for all the possible executions of a model (in our case the specific curriculum). This is often done by means of model checking techniques [12].

The curricula as we represent them are, actually, Kripke structures. Briefly, a Kripke structure identifies a set of states with a transition relation that allows passing from a state to another. In our case, the states contain the knowledge items that are owned at a certain moment. Since the domain is monotonic (as explained above we can assume that knowledge only grows), states will always contain *all* the competencies acquired up to that moment. The transition relation is given by the actions that are contained in the curriculum that is being checked.

It is possible to use the LTL logic to verify if a given formula holds starting from a state or if it holds for a set of states.

For example, in order to specify in the curricula model constraints on *what* to achieve, we can use the formula $\Diamond\alpha$, where $\Diamond$ is the eventually operator. Intuitively, such a formula expresses the fact that a set of knowledge elements will be acquired sooner or later. Moreover, constraints concerning *how* to achieve the educational objectives, such as "a knowledge element β cannot be acquired before the knowledge element α is acquired", can, for instance, be expressed by the LTL temporal formula $\neg\beta\ U\ \alpha$, where U is the *weak until* operator. Given a set of knowledge elements to be acquired, such constraints specify a partial ordering of the same elements.

2.3 Planning and Validation

Given a semantic annotation with preconditions and effects of the courses, classical planning techniques are exploited for creating *personalized curricula*, in the spirit of the work in [6, 7]. Intuitively the idea is that, given a repository of learning resources, which have been semantically annotated as described, the user expresses a *learning goal* as a set of *knowledge elements* he/she would like to acquire, and possibly also a set of already owned competencies. Then, the system applies planning to build a sequence of learning resources that, read in sequence, will allow him/her to achieve the goal.

The particular planning methodology that we implemented (see Section 3.3 for details) is a simple *depth-first forward planning* (an early prototype was presented in [3]), where actions cannot be applied more than once. The algorithm is simple:

1. Starting from the initial state, the set of *applicable* actions (those whose preconditions are contained in the current state) is identified.
2. One of such actions is selected and its application is simulated leading to a new state.
3. The new state is obtained by adding to the previous one the competencies supplied as effects of the selected action.
4. The procedure is repeated until either the goal is reached or a state is reached, in which no action can be applied and the learning goal is not satisfied.
5. In the latter situation, backtracking is applied to look for another solution.

The procedure will eventually end because the set of possible actions is finite and each is applied at most once. If the goal is achieved, the sequence of actions that label the transitions leading from the initial to the final state is returned as the resulting *curriculum*. If desired, the backtracking mechanism allows to collect a set of alternative solutions to present to the user.

Besides the capability of automatically building personalized curricula, it is also interesting to perform a set of verification tasks on curricula and curricula models. The simplest form of verification consists in *checking the soundness* of

curricula which are built by hand by users themselves, reflecting their own personal interests and needs. Of course, not all sequences which can be built starting from a set of learning resources are lawful. Learning dependencies, imposed by courses themselves in terms of preconditions and effects, must be respected. In other words, a course can appear at a certain point in a sequence only if it is *applicable* at that point, therefore, there are no *competency gaps*. These implicit "applicability constraints" capture precedences and dependencies that are innate to the nature of the taught concepts. In particular, it is important to verify that all the *competencies*, that are necessary to fully understand the contents, offered by a learning resource, are introduced or available before that learning resource is accessed. Usually, this verification, as stated in [13], is performed manually by the learning designer, with hardly any guidelines or support.

Given the interpretation of resources as actions, the verification of the *soundness of a curriculum*, w.r.t. the learning dependencies and the learning goal, can be interpreted as an *executability check* of the curriculum. Also in this case, the algorithm is simple:

1. Given an initial state, representing the knowledge available before the curriculum is attended, a simulation is executed, in which all the actions in the curriculum are (virtually) executed one after the other.
2. An action (representing a course) can be executed only if the current state contains all the concepts that are in the course precondition. Intuitively, it will be applied only if the student owns the notions that are required for understanding the topics of the course.
3. If, at a certain point, an action that should be applied is *not applicable* because some precondition does not hold, the verification fails and the reasons of such failure can be reported to the user.
4. Given that all the courses in the sequence can be applied, one after the other, the final state that is reached must be compared with the learning goal of the student: all the desired goal concepts must be achieved, so the corresponding knowledge elements must be contained in the final state.

This latter task actually corresponds to another basic form of verification, i.e. to check whether a (possibly hand-made) curriculum allows the *achievement of the desired learning goal*. These forms of basic verifications can be accomplished by the service described in Section 3.4.

Another interesting verification task consists in checking if a *personalized curriculum is valid w.r.t. a particular curricula model* or, following Brusilovski's terminology, checking if the curriculum is *compliant against the course design goals* [11]. Indeed, a personalized curriculum that is proved to be executable, cannot automatically be considered as being *valid* w.r.t. a particular *curricula model*. A curricula model, in fact, imposes further constraints on *what* to achieve and *how* achieving it. We will return to this kind of verification in Section 3.4.

3 Implementation in the Personal Reader Framework

The Personal Reader Framework has been developed with the aim of offering a uniform entry point for accessing the Semantic Web, and in particular Semantic Web Services. Indeed it offers an environment for designing, implementing and realizing Web content readers in a service-oriented approach, for a more detailed description, see [18] (`http://www.personal-reader.de/`).

In applications based on the Personal Reader Framework, a user can select and combine —plug together— which personalized support he or she wants to receive. The framework has already been used for developing Web Content Readers that present online material in an embedded context [10, 1, 17]. Besides

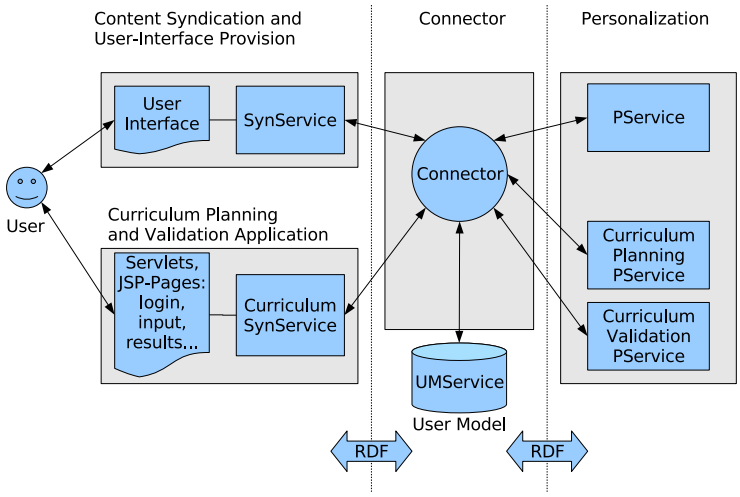

Fig. 1. Personal Reader Framework Overview

a user-interface, as shown in figure 1, a Personal Reader application consists of three types of *services*. *Personalization services* (PService) provide personalization functionalities: they deliver personalized recommendations for content, as requested by the user and obtained or extracted from the Semantic Web. *Syndication Services* (SynService) allow for some interoperability with the other services in the framework, e.g. for the discovery of the applications interfaces by a portal. The *Connector* is a single central instance responsible for all the communication between user interface and personalization services. It selects services based on their semantic description and on the requirements by the SynService. The Connector protects –by means of a public-key-infrastructure (PKI)– the communication among the involved parties. It also supports the customization and invocation of services and interacts with a user modelling service, called the *UMService*, which maintains a central user model.

3.1 Metadata Description of Courses

In order to create the corpus of courses, we started with information collected from an existing database of courses. We used the Lixto [9] tool to extract the needed data from the web-pages provided by the HIS-LSF (http://www.his.de/) system of the University of Hannover. This approach was chosen based on our experience with Lixto in the *Personal Publication Reader* [10] project, where we used Lixto for creating the publications database by crawling the publication pages of the project partners. The effort to adapt our existing tool for the new data source was only small. From the extracted metadata we created an RDF document, containing course names, course catalog identifier, semester, number of credit points, effects and preconditions, and the type of course, e.g. laboratory, seminar or regular course with examinations in the end, as illustrated in Figure 2.

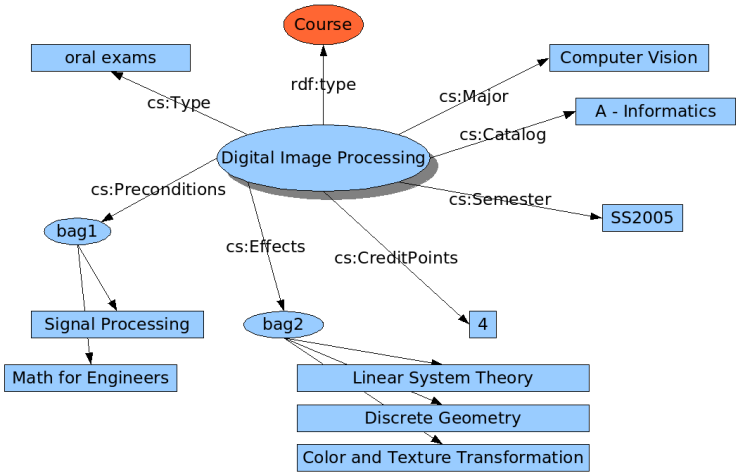

Fig. 2. An annotated course from the Hannover course database

The larger problem was that the quality of most of the information in the database turned out to be insufficient, mostly due to inconsistencies in the description of prerequisites and effects of the courses. Additionally the corpus was not annotated using a common set of terms, but authors and department secretaries used a slightly varying vocabulary for each of their course descriptions, instead of relying on a common classification system, like e.g. the ACM CCS for computer science.

As a consequence, we focussed only on a subset of the courses (computer science and engineering courses), and manually post-processed the data. Courses are annotated with prerequisites and effects, that can be seen as knowledge concepts or competences, i.e. ontology terms. After automatic extraction of effects

and preconditions, the collected terms were translated into proper English language, synonyms were removed and annotations were corrected where necessary. The resulting corpus had a total of 65 courses left, with 390 effects and 146 preconditions.

3.2 The User Interface and Syndication Service

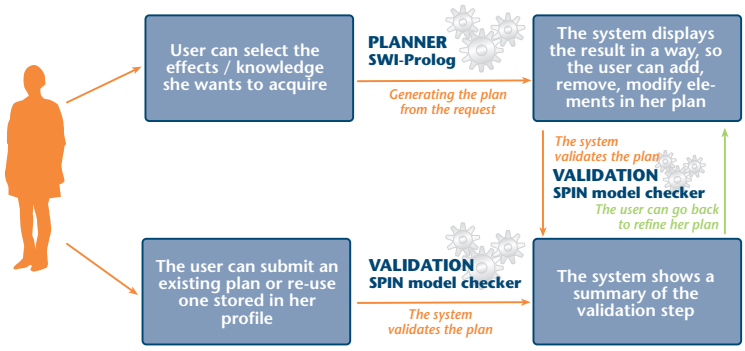

Fig. 3. The Actions supported by the User Interface

In our implementation, the user interface (see figure 3) is responsible for identifying the user, presenting the user an interface to select the knowledge she wants to acquire, and to display the results of the planning and validation step, allowing further refinement of created plans. The creation of curriculum sequences and the validation are implemented as two independent Personalization Services, the "Curriculum Planning PService", and the "Curriculum Validation PService". Because of the plug-and-play nature of the infrastructure, the two PServices can be used by other applications (SynServices) as well (Fig. 3). Also possible is that PServices, which provide additional planning and validation capabilities can be used in our application. The current and upcoming future implementations of the Curriculum Planning and Validation Prototype are available at `http://semweb2.kbs.uni-hannover.de:8080/plannersvc`.

3.3 The Curriculum Planning PService

In order to integrate the Planning Service as a plug-and-play personalization service in the Personal Reader architecture we worked at embedding the Prolog reasoner into a web service. Figure 4 gives an overview over the components in the current implementation. The web service implements the Personalization

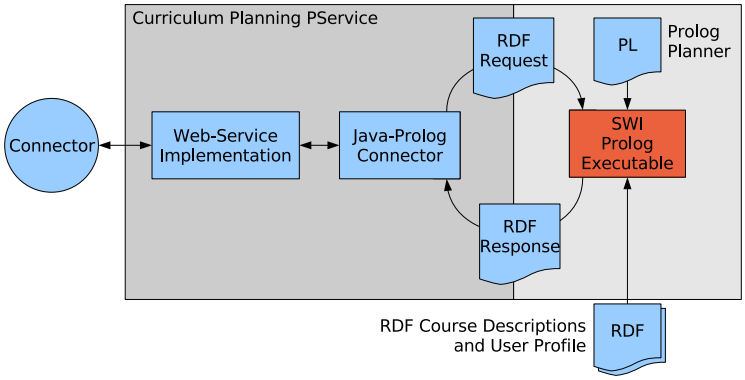

Fig. 4. Curriculum Planning Web Service

Service (*PService* [18]) interface, defined by the Personal Reader framework, which allows for the processing of RDF documents and for inquiring about the services capabilities. The *Java-to-Prolog Connector* runs the SWI-Prolog executable in a sub-process; essentially it passes the RDF document containing the request *as-is* to the Prolog system, and collects the results, already represented as RDF.

The curriculum planning task itself is accomplished by a reasoning engine, which has been implemented in SWI Prolog[3]. The interesting thing of using SWI Prolog is that it contains a semantic web library allowing to deal with RDF statements. Since all the inputs are sent to the reasoner in a *RDF request document*, it actually simplifies the process of interfacing the planner with the Personal Reader. In particular the request document contains: a) links to the RDF document containing the database of courses, annotated with metadata, b) a reference to the user's context c) the user's actual learning goal, i.e. a set of knowledge concepts that the user would like to acquire, and that are part of the *domain ontology* used for the semantic annotation of the actual courses. The reasoner can also deal with information about credits provided by the courses, when the user sets a credit constraint together with the learning goal.

Given a request, the reasoner runs the Prolog planning engine on the database of courses annotated with prerequisites and effects. The initial state is set by using information about the user's context, which is maintained by the User Modelling component of the PR. In fact such user's context includes information about what is considered as already learnt by the student (attended courses, learnt concepts) and such information is included in the request document. The Prolog planning engine has been implemented by using a classical depth-first search algorithm [22]. This algorithm is extremely simple to implement in declarative languages as Prolog.

[3] `http://www.swi-prolog.org/`

At the end of the process, a *RDF response document* is returned as an output. It contains a list of plans (sequences of courses) that fulfill the user's learning goals and profile. The maximum number of possible solutions can be set by the user in the request document. Notice that further information stored in the user profile is used at this stage for adapting the presentation of the solutions, here simple hints are used to *rank higher* those plans that include topics that the user has an expressed special interest in.

3.4 The Curriculum Validation PService

In order to verify if a curriculum is valid w.r.t. a curricula model, we adopt *model checking* techniques, by using SPIN. To check a curriculum with SPIN, this must be translated in the Promela language. *Competencies* are represented as *boolean variables*. In the beginning, only those variables that represent the initial knowledge of the student are true. *Courses* are implemented as actions that can modify the value of the variables. Since our application domain is monotonic, only those variables, whose value is false in the initial state, can be modified.

The Promela program consists of two processes: one is named *CurriculumVerification* and the other *UpdateState*. While the former contains a representation of the curriculum itself, and simulates its execution, the latter contains the code for updating the state (i.e. the set of competencies achieved so far) step by step along the simulation of the execution of the curriculum. The two processes communicate by means of two channels, *attend* and *feedback*. The notation *attend!courseName* represents the fact that the course with name *courseName* is to be attended. In this case the sender process is *CurriculumVerification* and the receiver is *UpdateState*. *UpdateState* will check the preconditions of the course in the current state and will send a feedback to *CurriculumVerification* after updating the state. On the other hand, the notation *feedback?feedbackMsg* represents the possibility for the process *Curriculum* of receiving a feedback of kind *feedbackMsg* from the process *UpdateState*.

Given these two processes, it is possible to perform a test, aimed at verifying the possible presence of competency gaps. This test is implemented as a *deadlock* verification: if the sequence is correct w.r.t. the action theory, no deadlock arises, otherwise a deadlock will be detected. The *curricula model* is to be supplied apart, as a set of temporal logic formulas, possibly obtained by an automatic translation process from a DCML representation. Notice that curricula can contain branching points. The branching points are encoded by either conditioned or non-deterministic *if*; each such *if* statement refers to a set of alternative courses (e.g. *languagesEnvironmentProg* and *programmingLanguages*). Depending on the course communicated by the channel *attend*, it updates the state. The process continues until the message *stop* is communicated. Then the learning goal is checked.

Let us see how to use the model checker to verify the *temporal constraints* that make a curricula model. Model checking is the algorithmic verification of the fact that a finite state system complies to its specification. In our case the

specification is given by the curricula model and consists of a set of temporal constraints, while the finite state system is the curriculum to be verified.

SPIN allows to specify and verify every kind of LTL formulas and it also allows to deal with curricula that at some points contain alternatives. This makes the system suitable to more realistic application scenarios. In fact, for what concerns curricula written by hand, users often do not have a clear mind and, thus, it is difficult for them to write a single sequence. In the case of curricula built by an automatic system, there are planners that are able to produce sets of alternative solutions gathered in a tree structure.

The following are examples of constraints, expressed as LTL formulas, that could be part of a curricula model:

(1) $\neg jdbc \,\mathsf{U}\, (sql \wedge relational\_algebra)$,
(2) $\neg op\_systems \,\mathsf{U}\, basis\_of\_prog$,
(3) $\neg basis\_of\_oo \,\mathsf{U}\, basis\_of\_prog$,
(4) $\Diamond basis\_of\_prog \supset \Diamond basis\_of\_java\_prog$,
(5) $\Diamond database$,
(6) $\Diamond web\_services$.

The first constraint means that before learning *jdbc* the student must own Knowledge about *sql* and about *relational algebra*. The following two constraints are of the same kind but involve different competencies. Constraint (4) means that if the student acquires knowledge about "basis of programming", he/she will also have knowledge about "basis of java programming" but the two events are not temporally related. Constraints (5) and (6) mean that soon or later knowledge about databases and web services must be acquired.

4 Conclusion, Further and Related Works

In this work we have described the current state of the integration of semantic personalization web services for Curriculum Planning and Validation within the Personal Reader Framework. The goal of personalization is to create sequences of courses that fit the specific context and the learning goal of individual students. Despite some manual post-processing for fixing inconsistencies, we used real information from the Hannover University database of courses for extracting the meta-data. Currently the courses are annotated also by meta-data concerning the schedule and location of courses, like for instance room-numbers, addresses and teaching hours. As a further development, it would be interesting to let our Curriculum Planning Service to make use also of such metadata in order to find a solution that fits the desires and the needs of the user in a more complete way.

The Curriculum Planning Service has been integrated as a new plug-and-play personalization service in the Personal Reader framework. In the current implementation, the learning goal corresponds to a set of hard constraints; that is to say that the planner returns only plans that satisfy them *all*. A different choice would be to consider the constraints given by the goal as *soft* constraints, and allow the return of plans which do satisfy the goal only partially. This

would be approapriate, for instance, in the case in which a student would like to acquire a range of competencies of interest but it is not possible to build, on top of a given repository of course descriptions, a curriculum for achieving them all. Nevertheless, it would be possible to build a curriculum for achieving *part* of them. In some circumstances, it would anyway be helpful for the student to receive this information as a feedback. Of course, in this case many questions arise, e.g. the issue of ranking the goals based on the actual interest of the requestor, so to know what can possibly be discarded and what is mandatory. From an implementation perspective, the spirit of the SOA infrustructure given to the Personal Reader is, indeed, meant to easily allow extensions by adding new Personalization Services. We can, therefore, think to develop and add a soft-goal planning service, to be used in these circumstances. The new planner would inherit the wrapping and interaction part from the current planning service but implement an algorithm like for instance [16].

The Curriculum Validation Service has been designed. An early prototype of the validation system based on the model checker SPIN has been developed [5] and is currently being embedded in the same framework. The choice of relying on SPIN, rather than developing a simpler and ad hoc checking system, is due to the need of rapidly developing a prototype. For this reason we have decided to rely on already exisiting and well-established technology. The engineering of the developed services should be tailored to the specific kinds of constraint that can be used to design the model. Analogous considerations can be done for the planning algorithm. The one that has been used is the simplest that can be thought of. Of course, there are many possible optimizations and extensions (e.g. the adoption of soft goals mentioned above) that could be done, and many algorithms are already available in the literature. Our choice has been motivated by the desire of quickly testing our ideas rather than developing a system thought for real use.

The Personal Reader Platform provides a natural framework for implementing a service-oriented approach to personalization in the Semantic Web, allowing to investigate how (semantic) web service technologies can provide a suitable infrastructure for building personalization applications, that consist of re-usable and interoperable personalization functionalities. The idea of taking a service oriented approach to personalization is quite new and was born within the personalization working group of the Network of Excellence REWERSE (Reasoning on the Web with Rules and Semantics, http://rewerse.net).

Writing curricula models directly in LTL is not an easy task for the user. For this reason, we have recently developed a graphical language, called DCML (Declarative Curricula Model Language) [8, 4], inspired by DecSerFlow, the Declarative Service Flow Language by van der Aalst and Pesic [23]. DCML allows to express the temporal relations between the times of acquisition of the concepts. The advantage of a graphical language is that *drawing*, rather than *writing*, constraints facilitates the user, who needs to represent curricula models, allowing a general overview of the relations which exist between concepts. At the same time, a rigorous and precise meaning is also given, due to the logic grounding of

the language. Moreover, in [4] we represent curricula as UML activity diagrams and include the possibility of handling the concurrent attending of courses. Also in this case curricula can be translated in Promela programs so that it becomes possible to perform all the kinds of verification that we have described.

DCML, besides being a graphical language, has also a textual representation. We are currently working at an integration of this new more sophisticated solution into the Personal Reader Framework by implementing an automatic system for translating DCML textual representations into LTL, for translating curricula (activity diagrams) in Promela, and then run the checks.

Another recent proposal for automatizing the competency gap verification is done in [21] where an analysis of pre- and post-requisite annotations of the Learning Objects (LO), representing the learning resources, is proposed. In this approach, whenever an error will be detected by the validation phase, a correction engine will be activated. This engine will use a "Correction Model" to produce suggestions for correcting the wrong curriculum, by means of a reasoning-by-cases approach. The suggestions will, then, be presented to the course developer, who is in charge to decide which ones to adopt (if any). Once a curriculum will have been corrected, it will have to be validated again, because the corrections might introduce new errors. Melia and Pahl's proposal is inspired by the CocoA system [11], that allows to perform the analysis and the consistency check of static web-based courses. Competency gaps are checked by a prerequisite checker for *linear courses*, simulating the process of teaching with an overlay student model. Pre- and post-requisites are represented by knowledge elements.

Acknowledgement This research has partially been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. `http://rewerse.net`).

References

1. F. Abel, I. Brunkhorst, N. Henze, D. Krause, K. Mushtaq, P. Nasirifar, and K. Tomaschweski. Personal reader agent: Personalized access to configurable web services. Technical report, Distributed Systems Institute, Semantic Web Group, University of Hannover, 2006.
2. Advanced Distributed Learning Network. SCORM: The sharable content object reference model, 2001. http://www.adlnet.org/Scorm/scorm.cfm.
3. M. Baldoni, C. Baroglio, I. Brunkhorst, N. Henze, E. Marengo, and V. Patti. A Personalization Service for Curriculum Planning. In E. Herder and D. Heckmann, editors, *Proc. of the 14th Workshop ABIS*, pages 17–20, Hildesheim, Germany, October 2006.
4. M. Baldoni, C. Baroglio, and E. Marengo. Curricula Modeling and Checking. In *Proc. of AI*IA 2007: Advances in Artificial Intelligence*, volume 4733 of *LNAI*, pages 471–482. Springer, 2007.
5. M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and L. Torasso. Verifying the compliance of personalized curricula to curricula models in the semantic web. In *Proc.*

of the Semantic Web Personalization Workshop, pages 53–62, Budva, Montenegro, 2006.
6. M. Baldoni, C. Baroglio, and V. Patti. Web-based adaptive tutoring: An approach based on logic agents and reasoning about actions. *Artificial Intelligence Review*, 1(22):3–39, 2004.
7. M. Baldoni, C. Baroglio, V. Patti, and L. Torasso. Reasoning about learning object metadata for adapting SCORM courseware. In L. Aroyo and C. Tasso, editors, *Int. Workshop on Engineering the Adaptive Web, EAW'04*, pages 4–13, 2004.
8. M. Baldoni and E. Marengo. Curriculum Model Checking: Declarative Representation and Verification of Properties. In *Proc. of 2nd Eur. Conf. EC-TEL*, volume 4753 of *LNCS*, pages 432–437. Springer, 2007.
9. R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *VLDB*, pages 119–128. Morgan Kaufmann, 2001.
10. R. Baumgartner, N. Henze, and M. Herzog. The personal publication reader: Illustrating web data extraction, personalization and reasoning for the semantic web. In *ESWC*, pages 515–530, 2005.
11. P. Brusilovsky and J. Vassileva. Course sequencing techniques for large-scale web-based education. *Int. J. Cont. Engineering Education and Lifelong learning*, 13(1/2):75–94, 2003.
12. O. E. M. Clarke and D. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 2001.
13. Juri L. De Coi, Eelco Herder, Arne Koesling, Christoph Lofi, Daniel Olmedilla, Odysseas Papapetrou, and Wolf Sibershi. A model for competence gap analysis. In *Proc. of WEBIST 2007*, 2007.
14. E. A. Emerson. Temporal and model logic. In *Handbook of Theoretical Computer Science*, volume B, pages 997–1072. Elsevier, 1990.
15. European Commission, Education and Training. The Bologna process. `http://ec.europa.eu/education/policies/educ/bologna/bologna_en.html`.
16. E. Giunchiglia and M. Maratea. SAT-based planning with minimal-♯actions plans and "soft" goals. In *Proc. of AI*IA 2007: Advances in Artificial Intelligence*, volume 4733 of *LNAI*. Springer, 2007.
17. N. Henze. Personal readers: Personalized learning object readers for the semantic web. In *12th International Conference on Artificial Intelligence in Education, AIED05*, Amsterdam, The Netherlands, 2005.
18. N. Henze and D. Krause. Personalized access to web services in the semantic web. In *The 3rd International Semantic Web User Interaction Workshop (SWUI, collocated with ISWC 2006*, November 2006.
19. IMSGlobal. Learning design specifications. Available at `http://www.imsglobal.org/learningdesign/`.
20. R. Koper and C. Tattersall. *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training*. Springer Verlag, 2005.
21. M. Melia and C. Pahl. Automatic Validation of Learning Object Compositions. In *Information Technology and Telecommunications Conference IT&T'2005: Doctoral Symposium*, Carlow, Ireland, 2006.
22. S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.
23. W. M. P. van der Aalst and M. Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. In Mario Bravetti and Gialuigi Zavattaro, editors, *Proc. of WS-FM*, LNCS, Vienna, September 2006. Springer.