Curriculum Model Checking: Declarative Representation and Verification of Properties

Matteo Baldoni and Elisa Marengo

Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
baldoni@di.unito.it,elisa.mrng@gmail.com

Abstract. When a curriculum is proposed, it is important to verify at least three aspects: that the curriculum allows the achievement of the user's learning goals, that the curriculum is compliant w.r.t. the course design goals, specified by the institution that offers it, and that the sequence of courses that defines the curriculum does not have competency gaps. In this work, we present a constrained-based representation for specifying the goals of "course design" and introduce a design graphical language, grounded into Linear Time Logic.

Keywords: formal model for curricula description, *model checking*, *verification of properties*, *competence gaps*.

1 Introduction and Motivations

As recently underlined by other authors, there is a strong relationship between the development of peer-to-peer, (web) service technologies and e-learning technologies [11,8]. The more learning resources are freely available through the Web, the more e-learning management systems (LMSs) should be able to take advantage from this richness: LMSs should offer the means for easily retrieving and assembling e-learning resources so to satisfy specific users' learning goals, similarly to how services are retrieved and composed [8]. As in a service composition it is necessary to verify that, at every point, all the information necessary to the subsequent invocations is available, in a learning domain, it is important to verify that all the *competencies*, i.e. the *knowledge*, necessary to fully understand a learning resource are introduced or available before that learning resource is accessed. The composition of learning resources, i.e. a *curriculum*, does not have to show any *competency gap*. Unfortunately, this verification, is usually performed *manually* by the designer, with hardly any guidelines or support [6].

In [11] an analysis of pre- and post-requisite annotations of the Learning Objects (LO), representing the learning resources, is proposed for automatizing the competency gap verification. A logic based validation engine can use these annotations to validate the LO composition. This proposal is inspired by the CocoA system [5], 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 concepts, elementary pieces of domain of knowledge.

Brusilovsky and Vassileva [5] sketch many other kinds of verification. In our opinion, two of them are particularly important: (a) verifying that the curriculum allows to achieve the users' *learning goals*, and (b) verifying that the curriculum is compliant against the *course design goals*. Verifying (a) is fundamental to guarantee that users will acquire the desired knowledge. At the same time, manually or automatically supplied curricula, developed to reach that learning goal, should match the design document, a curricula model, specified by the institution. Curricula models specify general rules for designing sequences of learning resources (courses) and can be interpreted as *constraints*. These constraints are to be expressed in terms of *concepts* and, in general, it is not possible to associate them directly to a learning resource, as instead is done for pre-requisites, because they express constraints on the acquisition of concepts, independently from the resources that supply them.

This work differs from previous work [4], where the authors presented an adaptive tutoring system, that exploits *reasoning about actions and changes* to plan and verify curricula. That approach was based on abstract representations, capturing the *structure* of a curriculum, and implemented as prolog-like clauses. A procedure-driven planning was applied to build personalized curricula. The advantage of such planning techniques is that the only curricula that are tried are the possible executions of the procedure itself, and this restricts considerably the search space of the planning process. In this context, we proposed also forms of verification: of competency gaps, of learning goal achievement, and of whether a curriculum, given by a user, is compliant to the *course design* goals. The use of procedure clauses is, however, limiting because they, besides having a *prescriptive* nature, pose very strong constraints on the sequencing of learning resources. Clauses represent what is "legal" and whatever sequence is not foreseen by the clauses is "illegal". However, in an open environment where resources are extremely various, they are added/removed dynamically, this approach becomes unfeasible.

For this reason it is appropriate to take another perspective and represent only those constraints which are strictly necessary, in a way that is inspired by the so called *social approach* proposed by Singh for describing communication protocols for multi-agent systems and service oriented architecture [12]. In this approach only the *obligations* are represented. In our application context, obligations capture relations among the times at which different competencies are to be acquired. The advantage of this representation is that we do not have to represent all that is legal but only those *necessary conditions* that characterize a legal solution. To make an example, by means of constraints we can request that a certain knowledge is acquired before some other knowledge, without expressing what else is to be done in between.

In this paper we present a constraint-based representation of curricula models. Constraints are expressed as formulas in a temporal logic (LTL, linear-time logic [7]) represented by means of a simple graphical language that we call DCML (*Declarative Curricula Model Language*). This kind of logic allows the verification of some properties of interest for all the possible executions of a model, which in our case corresponds to the specific curriculum.

2 DCML: A Declarative Curricula Model Language

In this section we describe our Declarative Curricula Model Language (DCML), a graphical language to represent the relations that can occur among concepts supplied by attending courses. DCML is inspired by DecSerFlow, the Declarative Service Flow Language to specify, enact, and monitor web service flows [13]. As such, DCML is grounded in Linear Temporal Logic (LTL) [7] and it allows a curricula model to be described in an easy way, with a rigorous and precise meaning given by the logic representation. LTL includes temporal operators such as *next-time* ($\bigcirc\varphi$, the formula φ holds in the immediately following state of the run), *eventually* ($\Diamond\varphi$, φ is guaranteed to eventually become true), *always* ($\Box\varphi$, the formula φ remains invariably true throughout a run), *until* ($\alpha \mathsf{U} \beta$, the formula α remains true until β). The set of LTL formulas obtained for a curricula model are, then, used to verify whether a curriculum will respect it [3]. As an example, Fig. 1 shows a curricula model expressed in DCML. Every box

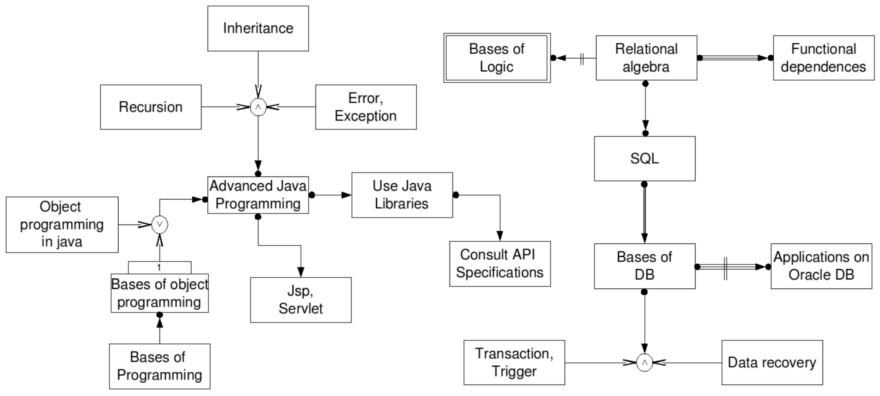

Fig. 1. An example of curricula model in DCML

contains at least one competency. Boxes/competencies are related by arrows, which represent (mainly) temporal constraints among the times at which they are to be acquired. Altogether the constraints describe a curricula model. Hereafter, we describe most of such elements.

The simplest kinds of constraint concern the *existence*, *absence*, or *possibility* of acquisition for a certain competency. The *existence constraint* imposes that a certain concept k must be acquired sooner or later. It captures the fact that a concept characterizes a curriculum, so a student cannot present a plan in which it does not appear. It is represented by the LTL formula $\Diamond k$, that is k must eventually become true. Similarly, a course designer can impose that a concept k must never appear in a curriculum. This is possible by means of the *absence constraint*. The LTL formula $\Box\neg k$ expresses this fact: it means that k cannot appear. On the diagram these two constraints are given by marking boxes with the "cardinality" of the concepts (1 for existence and 0 for absence). When both 0 and 1 appear on the same box, we have a *possibility constraint*. The corresponding LTL formula is $\Diamond k \vee \Box\neg k$. When no cardinality is expressed explicitly, possibility is assumed. The last constraint on concepts is represented by a double box, which means that a concept k must belong to the initial knowledge of the student. In other words, the simple logic formula k must hold in the initial state.

In DCML it is also possible to represent *Disjunctive Normal Form* (DNF) formulas as *conjunctions* and *disjunctions* of concepts. For lack of space, we do not describe the notation here, although an example can be seen in Fig. 1. The interested reader can find an extended version of this paper that is available in the home page of the authors.

Besides the representation of competencies and of constraints on competencies, DCML allows to represent *relations* among competencies. For simplicity, in the following presentation we will always relate simple competencies, although it is, of course, possible to connect DNF formulas.

Arrows ending with a little-ball, express the *before* temporal constraint between two concepts: a concept must be acquired *before* another one. This constraint can be used to express that, to understand a topic, some other knowledge is required. Notice that if the antecedent never becomes true, also the consequent must be invariably false. k_1 *before* k_2 corresponds to the LTL formula $\neg k_2 \;\mathsf{U}\; k_1$.

One can express that a concept must be acquired *immediately before* some other by means of a triple line arrow that ends with a little-ball. The constraint "k_1 *immediate before* k_2" imposes that k_1 is acquired before k_2 and that either k_2 is true in the next state (w.r.t. when k_1 is acquired) or it is never acquired. Immediate before is stronger than before because it imposes that two concepts have to be acquired in strict sequence. The LTL formula for *immediate before* is $\neg k_2 \;\mathsf{U}\; k_1 \wedge \Box(k_1 \supset (\bigcirc k_2 \vee \Box\neg k_2))$, that is k_1 *before* k_2 and whenever k_1 holds, either in the next state k_2 holds or k_2 never holds.

The *implication* relation specifies, instead, that if a certain concept holds, some other concept must be acquired sooner or later. The acquisition of the consequent is imposed by the truth value of the antecedent, but, in case this one is true, the implication does not specify when the consequent is to be achieved (it could be before, after or in the same state as the antecedent). k_1 *implies* k_2 is expressed by the LTL formula $\Diamond k_1 \supset \Diamond k_2$.

The *immediate implication* instead, specifies that the consequent must hold in the state right after the one in which the antecedent is acquired. This does not mean that it must be acquired in that state, but only that it cannot be acquired after. This is expressed by the LTL implication formula in conjunction with the constraint that whenever k_1 holds, k_2 holds in the next state: $\Diamond k_1 \supset \Diamond k_2 \wedge \Box(k_1 \supset \bigcirc k_2)$. *Implication* and *immediate implication* are graphically represented with an arrow that starts with a little-ball and with a triple line arrow that starts with a little-ball.

The last two kinds of temporal constraints are *succession* and *immediate succession* The *succession* relation specifies that if k_1 is acquired, afterwards k_2 is also achieved. Succession is expressed by the LTL formula $\Diamond k_1 \supset (\Diamond k_2 \wedge (\neg k_2 \;\mathsf{U}\; k_1))$. While in the *before* relation, when the antecedent is never acquired also the consequent must be false, in the *succession* relation this is not relevant. This behaviour is due to the fact that the *succession* specifies a condition of the kind: *if* k_1 *then* k_2. The *before*, instead, represents a constraint without any conditional premise. The fact that the consequent must be acquired *after* the antecedent differentiates *implication* from *succession*.

The *immediate succession* imposes that the acquisition of the consequent must happen either in the same state the antecedent is acquired or in the state immediately after (not before nor later). The immediate succession is expressed by the LTL formula: $\Diamond k_1 \supset (\Diamond k_2 \wedge (\neg k_2 \;\mathsf{U}\; k_1)) \wedge \Box(k_1 \supset \bigcirc k_2)$. The representation of (*immediate*) *succession*, see Fig. 1, is an (triple) arrow that starts and ends with a little-ball.

The graphical notations for "negative relations" is very intuitive: two vertical lines break the arrow that represents the constraint. Some examples are shown in Fig. 1.

k_1 *not before* k_2 specifies that the concept k_1 cannot be acquired before or in the same state of the concept k_2. The LTL formula is $\neg k_1 \ \mathsf{U}\ (k_2 \wedge \neg k_1)$. Notice that this is not obtained by simply negating the before relation but it is weaker because the negation would impose the acquisition of the concepts specified as consequents, the *not before* does not. The *not immediate before* is translated exactly in the same way of the *not before*. Indeed, it is a special case of *not before*. This happens because the acquired knowledge cannot be forgotten.

By means of k_1 *not implies* k_2 we express that the acquisition of the concept k_1 implies that k_2 will never be acquired. We express this by the LTL formula $\Diamond k_1 \supset \Box \neg k_2$. Again, we choose to use a weaker formula than the natural negation of the implication relation, that is $\Diamond k_1 \wedge \Box \neg k_2$. k_1 *not immediate implies* k_2 constraint imposes that when the concept k_1 is acquired, in the immediately subsequent state, the concept k_2 must be false. Afterwards, the truth value of k_2 does not matter (it is weaker than $\neg(k_1$ *immediate implies* $k_2))$. The corresponding LTL formula is $\Diamond k_1 \supset (\Box \neg k_2 \vee \Diamond(k_1 \wedge \bigcirc \neg k_2))$.

The *not succession*, and the *not immediate succession* are weaker versions of, respectively, negation of succession and of immediate succession. The first one imposes that a concept cannot be acquired after another. This means that it could be acquired before, or it will always be false. The LTL formula is $\Diamond k_1 \supset (\Box \neg k_2 \vee k_1$ *not before* $k_2)$. The second imposes that if a concept is acquired in a certain state, in the state that follows another concept must be false: $\Diamond k_1 \supset (\Box \neg k_2 \vee k_1$ *not before* $k_2 \vee \Diamond(k_1 \wedge \bigcirc \neg k_2))$.

3 Conclusions

The presented work is an evolution of earlier works [2,4]. In those works, we semantically annotated learning objects with the aim of building compositions of new learning objects, based on the user's learning goals and by exploiting planning techniques. That proposal was based on a different approach, that relied on the experience of the authors in the use of techniques for reasoning about actions and changes. Of course, the new proposal, presented in this paper, can be applied also to learning objects, given a semantic annotation of theirs, as introduced in the cited works. In [1] we discuss the integration, into the Personal Reader Framework [9], of a verification web service that implements the explained techniques.

In particular, in this work we have introduced a graphical language to describe temporal constraints posed on the acquisition of competencies (supplied by courses). In the extended version, that is available on-line, we show how to use UML activity diagrams to specify sets of curricula, and we show how to translate them into Promela programs. Such programs can be used by the SPIN model checker [10] to verify whether the curriculum respects the DCML model. Model checking can also be applied for checking the achievement of the user's learning goals and the presence of competency gaps.

In [3] we extend the current proposal so as to include a representation of the *proficiency level* at which a competency is owned or supplied, as suggested in [6]. The key idea is to associate to each competency a variable, having the same name as the competency, which can be assigned natural numbers as values. The value denotes the proficiency level; zero means absence of knowledge. The next step will be to give a structure to competencies, e.g. by defining a proper ontology, for allowing more flexible forms of reasoning and verification.

We are currently working on an automatic translation from a textual representation of DCML curricula models into the corresponding set of LTL formulas and from a textual representation of an activity diagram, that describes a curriculum (comprehensive of the description of all courses involved with their preconditions and effects), into the corresponding Promela program. We are also going to realize a graphical tool to define curricula models by means of DCML.

Acknowledgements. The authors would like to thank Cristina Baroglio, and also Viviana Patti and Ingo Brunkhorst, for the helpful discussions. 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), and it has also been supported by MIUR PRIN 2005 "Specification and verification of agent interaction protocols" national project.

References

1. Baldoni, M., Baroglio, C., Brunkhorst, I., Marengo, E., Patti, V.: Curriculum Sequencing and Validation: Integration in a Service-Oriented Architecture. In: Proc. of EC-TEL 2007 (2007)
2. Baldoni, M., Baroglio, C., Henze, N.: Personalization for the Semantic Web. In: Eisinger, N., Małuszyński, J. (eds.) Reasoning Web. LNCS, vol. 3564, pp. 173–212. Springer, Heidelberg (2005)
3. Baldoni, M., Baroglio, C., Marengo, E.: Curricula Modeling and Checking. In: Proc. of AI*IA 2007, Rome, Italy. LNCS (LNAI), Springer, Heidelberg (2007)
4. Baldoni, M., Baroglio, C., Patti, V.: Web-based adaptive tutoring: an approach based on logic agents and reasoning about actions. Artificial Intelligence Review 22(1), 3–39 (2004)
5. Brusilovsky, P., Vassileva, J.: Course sequencing techniques for large-scale web-based education. Int. J. Cont. Engineering Education and Lifelong learning 13(1/2), 75–94 (2003)
6. De Coi, J.L., Herder, E., Koesling, A., Lofi, C., Olmedilla, D., Papapetrou, O., Sibershi, W.: A model for competence gap analysis. In: Proc. of WEBIST 2007, INSTICC Press (2007)
7. Emerson, E.A.: Temporal and model logic. Handbook of Theoretical Computer Science B, 997–1072 (1990)
8. Farrell, R., Liburd, S.D., Thomas, J.C.: Dynamic assebly of learning objects. In: Proc. of WWW 2004, New York, USA (2004)
9. Henze, N., Krause, D.: Personalized access to web services in the semantic web. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) ISWC 2006. LNCS, vol. 4273, Springer, Heidelberg (2006)
10. Holzmann, G.J.: The SPIN Model Checker. Addison-Wesley, Reading (2003)
11. Melia, M., Pahl, C.: Automatic Validation of Learning Object Compositions. In: Proc. of IT&T'2005: Doctoral Symposium, Carlow, Ireland (2006)
12. Singh, M.P.: Agent communication languages: Rethinking the principles. IEEE Computer 31(12), 40–47 (1998)
13. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, Springer, Heidelberg (2006)