An Integrated Architecture for Automatic Course Generation

Nicola Capuano[1,2], Matteo Gaeta[1,2], Alessandro Micarelli[1,3], Enver Sangineto[1]
[1]*CRMPA Centro di Ricerca in Matematica Pura ed Applicata*
Università degli Studi di Salerno - DIIMA
Via Ponte Don Melillo - 84084 Fisciano (SA), Italia
e-mail: {niccap, gaeta}@crmpa.it, {micarel, sanginet}@dia.uniroma3.it
[2]*Dipartimento di Ingegneria dell'Informazione e Matematica Applicata*
Università degli Studi di Salerno
Via Ponte Don Melillo - 84084 Fisciano (SA), Italia
[3]*Dipartimento di Informatica e Automazione*
Università degli Studi ROMA TRE
Via della Vasca Navale, 79 - 00146 Roma, Italia

Abstract

In this paper we propose a complete architecture of an automatic computer based educational system which exploits tools and methodologies taken from various Artificial Intelligence areas. We use a smart description of knowledge (concerning both the didactic domain knowledge and the student model) and inference mechanisms to achieve an efficient and reliable planning of the student course. Finally, we focused our attention also on standardization issues to allow portability to different e-learning platforms of all the didactic material

1. Introduction

In this paper we propose an integrated architecture for a computer-based learning system named LIA (Learning Intelligent Advisor). LIA is part of a bigger project (funded by EC under the 5th Framework Program – Information Society Technologies) named m-learning [1] whose objective consists in the development, the experimentation and the evaluation of product and service prototypes for the dissemination of didactical micro-modules through mobile technologies to promote the so-called "lifelong learning". LIA is the "intelligent" engine of the m-learning Course Management System able to offer automatic user assessment and course customization features. Such "intelligent" functionalities are realized by means of Artificial Intelligence techniques often applied in the Intelligent Tutoring Systems. As we will deepen in the following sections, LIA is composed of three main modules: the didactic knowledge (described in Section 2), the present student state (or *student model*, described in Section 3) and some planning procedures (Section 4) which are able to automatically create a course satisfying all the student learning requirements taking into account both her/him present knowledge and learning preferences.

2. The didactic knowledge representation

LIA didactic knowledge is represented through different abstraction levels. The lowest level is composed by *Learning Objects*. A Learning Object (LO) is defined [2] as any entity which can be used, re-used or referenced during technology-supported learning. In our case, a Learning Object is a logical container that represents an atomic Web-deliverable resource such as a Lesson (an HTML page), a Simulation (a Java applet), a Test (an HTML page with an evaluating form) and each kind of Web-deliverable object.
Learning Objects must be indexed in order to let LIA know what each one of them is about and how they can be used during the learning process. This is done by means of a second abstraction representation level (*Metadata*). A Metadata is a collection of attributes about a Learning Object (LO) describing some features such as its type (text, simulation, slide, questionnaire, ...), the required educational level (high school, university, ...), the language, the interactivity level and so on... Several organizations such as IEEE, EDUCOM, etc. are focusing their attention on the creation of Metadata standards specifying the syntax and the semantics of the attribute declarations. In LIA we adopted the *IMS standard* [3] of the *IMS Global Learning Consortium*.
Finally, a third abstraction level (called *Ontology*) is used to represent *Domain Concepts* and their relations. A Domain Concept (DC) is a concept belonging to the described didactic domain and can be possibly explained by one or more LOs. For example, in the *Math* domain, we can have concepts such as "Mathematical Analysis" or the (sub-) concepts "Limits", "Derivatives", ..., which,

in turn, can be explained by the LOs: "Introduction to Limits" (an HTML text), "Lesson 1: Limits" (slides), "Lesson 2: Derivatives" (slides), and so on. Furthermore, a DC can be a special LIA event (a *Milestone*) used to receive feedback from the student. A Milestone is associated with one or more LOs (of type *test*), and its use will be better explained in Section 4.

Typical relations among concepts are: *IsPart_of(Limits, Mathematical Analysis)* or *Requires(Derivatives, Limits)*, to indicate, respectively, a hierarchical relationship and a constraint on the learning order of two concepts. DCs and their relations are described in LIA by using a standard Ontology description language for Web resources called SHOE (Simple HTML Ontology Extensions, [4]).

In the following we give a more formal description of the LIA admissible relation types linking DCs:

- *IsPart_of(x,y)* means that *x* is a component of *y*.
- *Requires(x,y)* means that the *x* needs of *y* as a pre-requisite. This relation poses a constraint on the delivery order of the DCs to the student.
- *Suggested_Order(x,y)* means that it is *preferable* to learn *x* and *y* in this order. Note that also this relation poses a constraint on the DCs' order but now it is not necessary to learn *y* if we are interested only in *x*.
- *IsTested_by(x,ML)* means that *ML* (a Milestone) concerns the verification of the concept *x*.

All the proposed relations have a binary ariety, thus they can be easily represented by arcs in a graph data structure (where each node represents a DC). Besides the above relationships among DCs, we need of a link between the Ontology and the Metadata levels:

- *Explained_by(d,l)* means that the DC *d* can be explained by means of the LO which is described by the metadata *l*. Note that *d* can be a Milestone and *l* a test.

Finally, each metadata directly refers to the LO it describes. Figure 1 shows an example of knowledge representation. We used the following abbreviations: L = Limits, D = Derivatives, I = Integrals, S = Series, B = IsPart_of, R = Requires, SO = SuggestedOrder, E = Explained_by, LO_i = the metadata describing the i-th Learning Object. Note a LO can explain one or more DCs and each DC can be explained by more than one LOs. Indeed, the semantics of *Explained_by(x,y)* poses that *y* is enough to completely explain *x*. If *Explained_by(x,y)* and *Explained_by(x,z)* ($z \neq y$) then we intend that *z* and *y* are mutually exclusive. Moreover, if *Explained_by(x,y)* and *Explained_by(z,y)* then *y* contains enough contents to explain both DCs *x* and *z*. Finally, the figure does not explicitly show the lowest knowledge representation level because each LO metadata directly refers to its indexed resource in a unique and trivial way.

In order to simplify, since now on we indicate with the term "Learning Object" (shortly, LO), the structure given by the union of a learning resource and its metadata representation.

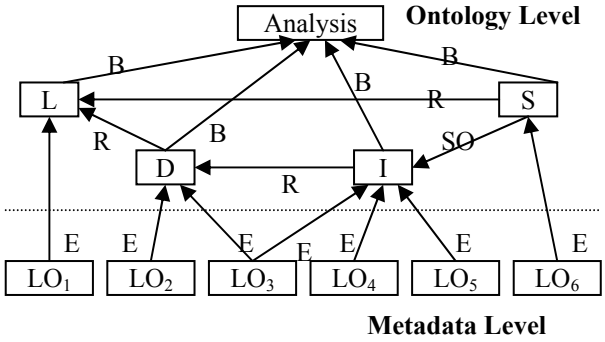

Figure 1. An example of knowledge representation belonging to the didactic domain "Math".

Before to conclude this section, we provide a last definition which will be useful in Section 4. A concept *x* is named *atomic* if there is no concept *y* s.t. *IsPart_of(y, x)*. It is important to note that only atomic DCs are linked to LOs by means of the relation *Explained_by*. Indeed, a not atomic concept *x* is composed of sub-concepts (e.g., *y*, *z*) and the learning of *x* is equivalent to the learning of its components *y* and *z*.

3. The student model

The aim of the LIA student model is to collect information about the student *Cognitive State* and *Learning Preferences*. We use the former to describe the knowledge degree achieved by each student about every DC. This evaluation regards both previously acquired student knowledge and skills learnt using our proposed e-learning platform. Moreover, the Learning Preferences module groups information about the student perceptive capabilities, i.e., the preferred types of resources for the specific student.

Cognitive State and Learning Preferences describe the student present state by means of two different sets of *atomic facts* associated with a fuzzy truth degree. More precisely, we adopt a couple of fuzzy values given by:

Evaluation = <*Degree, Reliability*>. (1)

Evaluation is a pair of fuzzy values ranging in $[0, 1]^2$. They indicate, respectively, the degree of knowledge/preference the student has shown concerning the associated *fact*, and the reliability degree that the system has about this evaluation. For example, if a DC is associated to a value of *Evaluation* = <*0.8, 0.6*>, this indicates that LIA believes the student knows quite much

about the related DC (0.8), and estimates the reliability of this statement about her/him knowledge with a certainty degree of 0.6.

The values of each *Evaluation* are continuously and automatically updated by LIA in the following manner. During the learning process the planning module fixes some Milestones in which the student is evaluated exploiting suitable tests. Student answers allow to directly estimate and modify her/his Cognitive State. About Learning Preferences, the system matches them with the Metadata describing the LOs delivered to the student, taking into account the test scores she/he has obtained with the Milestone test. Indeed, at each Milestone a new Evaluation is computed for each DC involved in the student performed activities. LIA can then evaluate the pedagogical effectiveness of the LO typologies by exploiting both the variation between concept evaluations and the Metadata information about delivered LOs between couples of subsequent Milestones.

In the following two sub-sections we describe more in details how the Cognitive State and the Learning Preferences of each student are represented in LIA.

3.1. The student Cognitive State

The LIA Cognitive State is composed of a set of student *beliefs* representing "the system knowledge about each student knowledge". If the Ontology (see Section 2) is composed by n DCs ($d_1,\ldots,d_n$), then the Cognitive State of a given student is represented by the set:

$Beliefs = \{B_1, B_2, \ldots, B_n\}$,

each B_i being a *belief* so defined:

$B_i = <d_i, Evaluation_i>$,

where d_i is the i-th Ontology DC, while $Evaluation_i$ is a pair of fuzzy values as defined in (1). If $Evaluation_i = <Degree_i, Reliability_i>$, then B_i associates the fact: "the student knows the concept d_i" with the truth fuzzy value $Degree_i$ and the reliability fuzzy value $Reliability_i$. All the beliefs of the set *Beliefs* are assumed to be in and. Finally, we remark that all beliefs are represented using or extending the data structures of the *Learner Information Packaging* (LIP) standard [5].

3.2. The student Learning Preferences

To evaluate the student resource preferences, we focalise our attention on some LO attributes contained into a few fields of the *Educational* Metadata category of the *IMS* learning standard. They refer to generic features of a LO, such as its *language*, *context* (the student educational level), *age range*, *typical learning time*, *interactivity level*, *learning resource type*, *difficulty* and so on. To maintain a descriptive uniformity with the Cognitive State, each feature is described by an atomic *fact* associated with a pair of fuzzy values of type Evaluation as defined in (1). Thus, if we take into account m features ($f_1, \ldots, f_m$), the student Learning Preferences are represented by the following set:

$LP = \{P_1, P_2, \ldots, P_m\}$,

each P_i being a *preference* so defined:

$P_i = <eq_i, Evaluation_i>$,

where eq_i is an atomic fact with the following syntax:

$f_i = v_i$.

f_i is an attribute name while v_i is an admissible value for f_i in the IMS Metadata. Let us look to the example below:

$<Learning\_Resource\_Type = Text, <0.7, 0.2>>$. (2)

(2) means that the student likes textual LOs (with a truth fuzzy value of 0.7), but the system up to now has collected very few cues supporting this statement (indeed, the reliability value, 0.2, is very low).

Hence, when the fuzzy values of $Degree_i$ and $Reliability_i$ of $Evaluation_i$ are enough high, we can match the feature name (f_i) and its value (v_i) with LOs' Metadata to choose the most suitable for the present student (see Section 4).

Finally, all the preferences of the set *LP* are represented using or extending the data structures of the LIP standard. Once again, let us underline that Learning Objects' Metadata ("this LO is of type text"), represented by means of the IMS standard, are conceptually different from the student Learning Preferences ("the student prefers LOs of type text"), represented with LIP. They only share the same vocabulary for attribute names and values to facilitate the matching phase.

4. The planning mechanisms

In this section we describe the steps needed to perform automatic *Presentation* generation in LIA. A Presentation is a list of LOs delivered to the student to entirely cope her/him learning requirements. The student submits to LIA a list (*TargetC*) of Target Concepts. A *Target Concept* is a DC belonging to the student learning objectives. Given *TargetC* as input, LIA builds a Presentation, namely a list of LOs which satisfy all the *TargetC* DCs taking into account the student's present state (given by the union of the facts belonging to the Cognitive State and the Learning Preferences).
More precisely, a Presentation *PR* is an ordered list of LOs ($PR = \{l_1,\ldots,l_n\}$) with the following properties:

1. The union of the LOs ($\bigcup_{i=1,\ldots,n} l_i$) of *PR* is sufficient to explain to the student all the Target Concepts belonging to *TargetC*.
2. For each l_i, $l_j \in PR$, **if** : *Explained_by*(d_1, l_i) **and** *Explained_by*(d_2, l_j) **and** $d_1 \prec d_2$, **then** $i<j$, where the partial order relation $\prec$ between DCs is recursively defined in the following manner:
   - **if** *Requires(x, y)* **then** $y \prec x$
   - **if** *SuggestedOrder (x, y)* **then** $x \prec y$
   - **if** *IsTested_by (x, y)* **then** $x \prec y$
   - **if** *IsPart_of (x, z)* **and** *IsPart_of (y, w)* **and** $z \prec w$ **then** $x \prec y \wedge x \prec w \wedge z \prec y$.
3. *PR* meets the Learning Preferences of the student present state and does not include any DC already known by the student as asserted in her/him present Cognitive State.

While Points 1 and 3 of the above definition are self-explaining, Point 2 needs some remarks. The relations among DCs defined in Section 2 pose a partial order on the elements of a didactic domain. As a consequence of this, LOs belonging to a same Presentation have to respect this partial order. If, for instance, a Presentation contains l_i and l_j, which explain, respectively, the concepts Derivatives and Limits, then l_j has to precede l_i. The same situation happens when l_i and l_j explain DCs not directly linked to each other by an order relation (*Requires*, *SuggestedOrder* or *IsTested_by*) but components of DCs directly linked by an order relation.

In the following two subsections we show, respectively, the general planning algorithms needed to generate a partial ordered Presentation satisfying Points 1 and 3, and the proposed linearization procedure to totally order a partial ordered Presentation according to Point 2.

4.1. Automatic Presentation generation

The Presentation generation algorithm collects in a Concepts' list named *AtomicList* all those *atomic* DCs which can be reached starting from the Target Concepts and following the links *IsPart_of*, *Requires* and *IsTested_by*. Then *AtomicList* is subsequently linearized (see the next subsection). Finally, for each of the *AtomicList* DCs, the algorithm looks for the LO whose Metadata best match the student Learning Preferences.

Presentation_Generation Algorithm.

Input: *TargetC*: DC list, *CS*: Cognitive State, *LP*: Learning Preferences. **Output:** LO list.

1. Check the Ontology consistence.
2. *Q*:= *TargetC*, *AtomicList*:= Nil, *PR*:= Nil.
3. For each $x \in Q$ s.t. $\neg$ *Known*(*x, CS*), do:
   a. If [*IsPart_of(y,x)* or *Requires(x,y)* or *IsTested_by*(*x,y*)] and $y \notin Q$, then insert *y* in *Q*.
   b. If $\neg \exists y$ s.t. *IsPart_of(y,x)*, then insert *x* in *AtomicList*.
1. *AtomicList*:= *Sort*(*AtomicList*, *TargetC*).
2. For each $x \in$ *AtomicList* do:
   a. Let *LOList* be the list of all the LOs *l* s.t. *Explained_by(x, l)* and *Consistent(l, LP)*.
   b. *BLO*:= Choose_the_best_of (*LOList,LP*).
   c. Insert *BLO* in *PR*.
   d. For each *x'* $\in$ *AtomicList* s.t. *Explained_by(x', BLO)*, delete *x'* from *AtomicList*.
1. Return *PR*.

Line 1 of the *Presentation_Generation* algorithm checks if the Ontology is consistent. This is done to avoid loops in the operations performed in Lines 3 and 4. An Ontology is inconsistent when, for example, $x \prec y$ and $y \prec z$ and $z \prec x$. The check is easily realized looking for a loop in the oriented graph representing the Ontology. For each Target Concept, in Lines 3.a we recursively collect all the DCs needed to learn it. In Line 3.b, a DC is added to the list of atomic concepts if it has not sub-components. Line 4 linerizes the atomic concept list just obtained (see the next subsection).

In Line 5, for each DC *x* of the atomic list, we choose a LO among all those LOs able to explain *x* (let us call them *LOList(x)*). It is important to note that the choice is local to *LOList(x)*. Indeed, if we look to the example of Figure 1, a LO can be linked to two or more DCs. For instance, LO_3 explains both *Derivatives* and *Integrals*, thus, if in Line 5.b the function:

Choose_the_best_of (LOList(Derivatives),LP) (3)

returns LO_3, then in Line 5.d *Integrals* is excluded by *Q*. Nevertheless, in Line 5.b we can have $BLO = LO_2$ because the function *Choose_the_best_of* depends only on *LOList(Derivatives)* and on *LP*. In other words, LO_2 could be judged better filling the student preferences with respect to LO_3, without taking into account any global optimization or minimization of the final LO list. The reason of this choice arises from the fact that a global optimization would lead to a combinatorial explosion, while we generally have only very few LOs linked to more then one DC.

Finally, the functions *Known*, *Consistent* and *Choose_the_best_of* are easily realized comparing DCs or LOs with the Cognitive State or Learning Preferences' facts. Depending on the function, we take into account in various ways only the *Degree* value of those facts whose *Reliability* value is over a pre-fixed threshold.

If the used lists (*Q, AtomicList, PR*, ...) are realized with a stack (or a queue) we have that the elements' insertion and deleting operations have a constant computational cost. Furthermore, if we associate to each DC and LO a Boolean flag, we can mark them when visited the first time (at list insertion). In this way we can be sure that no DC or LO is twofold visited for each algorithm execution. As a consequence of this, the computational cost of the Lines 1,2,3 and 5 is *O(n)*, where *n* indicates the number of DCs and LOs of the didactic domain.

4.2. The linearization phase

The idea behind the linearization algorithm is the following. Starting from the Target Concept list, we visit all the graph representing the Ontology, following all those relations which pose ordering constraints. While in the *Presentation_Generation* algorithm we marked a node the first time we visit it, now we do not use any Boolean flag. Indeed, we allow the same node to be visited by the linearization algorithm all the times it is reachable by a link chain. The reason of this choice is the following. Imagine to have these relations among DCs: *SO(x,y)*, *SO(x,z)*, *SO(y,w)* and *SO(z,w)* (where *SO* is an abbreviation for *SuggestedOrder*). Suppose now to visit the Ontology starting from *x*. If we perform a depth-first visit, from *x* we go to *y* (and we put *z* in the stack), then, from *y* we go to *w*, afterwards, from *z* (extracted from the stack) we try to go once more on *w*. If we do not allow re-visiting *w*, the resulting order is: {*x,y,w,z*}, which violets the constraint *SO(z,w)*. Conversely, if we have: *SO(x,y)*, *SO*(x,z), *SO(y,w)* and *SO(w,z)* and a breadth-first visit, we obtain, for example, {*x,y,z,w*}, which violets *SO(w,z)*. Vice versa, in the proposed algorithm, we adopt a depth-first visit but we allow the node *w* to be visited again after *z* extraction, thus obtaining: {*x,y,z,w*}, which is a correct linearization of the first example' s constraints.

Sort Algorithm.

Input : *AtomicList*, *TargetC* (DC lists). **Output:**DC list.

1. *max*:= 1, *Q*:= *TargetC*.
2. Until *Q* is not empty do:
   a. *x*:= *Pop*(*Q*), *order(x)*:= *max*, *max*:=*max* + 1.
   b. For each *y* s.t. *IsPart_of(y, x), Push(Q, y)*.
   c. For each *y* s.t. *Requires(x, y)* or *SuggestedOrder(y, x)* or *IsTested_by(y, x)*, *Push(Q, y)*.
1. Sort all the elements *x* of *AtomicList* according with the values of *order(x)* and following a *descendent* order.
2. Return *AtomicList*.

If *Q* is realized as a stack data structure, it ensures that, in Line 3.b, the components of a DC *x* are visited before any other DC *y* possibly related to *x* by any ordering relation (Line 3.c). Indeed, if *Requires(x,y)* and *IsPart_of(z,x)*, then we must have: $z \prec y$.

Line 2 totally orders the portion of the Ontology connected to the original Target Concepts (let us call it *Ontology(TargetC)*). Hence, each DC *x* belonging to *Ontology(TargetC)* in Line 3 is attached to an integer label (*order(x)*) which gives its position in a total order set. The more is *order(x)* with respect to *order(y)*, the less is the position of *x* with respect to *y* in the order. We now only have to select, among *Ontology(TargetC)*, the DCs previously chosen by the *Presentation_Generation* algorithm (and stored in *AtomicList*). Indeed, in Line 3 we use a common sorting algorithm to sort the *AtomicList* elements taking into account their *order* label in a *descending* order and we obtain the requested linearization.

If *n* is the number of DCs of *Ontology(TargetC)*, Line 3 is *O(nlog(n))*; while the whole procedure can be shown to be $O(n^2)$. We can then assert that overall computational cost of the *Presentation_Generation* algorithm is $O(n + n^2) = O(n^2)$, which is a good result, especially if we think that, generally *n* is never greater than 200.

5. Conclusions

In this paper we have presented the system LIA which realizes the intelligent functionalities of an automatic Course Management System developed in the European Project "m-learning". A prototype version of LIA has been implemented and is currently undergoing to significant experimental campaigns in the framework of the m-learning project. Early results confirm the correctness and reliability of the architecture and its very low-time consuming.

6. References

[1] http://www.m-learning.org/

[2] N. Capuano, M. De Santo, M. Marsella, M. Molinara, S. Salerno. *Personalised Intelligent Training on the Web: A Multi Agent Approach.* Electronic Business and Education, Recent Advances in Internet Infrastructures, Kluwer: Multimedia Systems And Applications Series, vol. 20, chap. 5, 2001.

[3] http://www.imsproject.org/enterprise

[4] **Heflin and J. Hendler**. *Searching the Web with SHOE*. In AAAI-2000 Workshop on AI for Web Search. 2000.

[5] http://www.imsproject.com/profiles/lipinfo01.html).