

Translating e-learning Flow-Oriented Activity Sequencing Descriptions into Rule-based Designs

Iván Martínez-Ortiz, José Luis Sierra, Baltasar Fernández-Manjón
Dpto. Ingeniería del Software e Inteligencia Artificial. Fac. Informática. Universidad Complutense, Madrid, Spain

Abstract

In this paper, we describe how to automatically translate e-learning flow-oriented activity sequences into rule-based designs, such as those supported by the “de-facto” e-learning modeling standard: the IMS Learning Design specification. Our aim is that instructors model their educational designs using a user-friendly visual notation. Then these designs can be automatically exported into standardized and interoperable representations, which can be interchanged with / deployed in a plethora of heterogeneous Learning Management Systems and tools. This approach has been implemented in e-LD, an authoring system which supports the authoring and refactoring of IMS Learning Designs using a flow-oriented visual syntax.

Keywords: Educational Modeling Languages, Activity Sequencing, Learning Design, Graphical authoring

1. Introduction

The design of e-learning courses is more than just creating the content because this design is supposed to identify and characterize many different and interwoven aspects [3]: e.g. suitable pieces of content, the activities to perform during the course, the roles played by the different actors involved in these activities, the services that support the learning process, etc. All these aspects are integral parts of the teaching and learning *methods* used in the courses. However, the incorporation of these methods into current Learning Management Systems (LMS) is a difficult task, since it implies close collaboration between two very different communities: *Technicians* and *Instructors*. While technicians are experts in hardware and software, they have little knowledge of the educational method or the domains covered in the courses. On the other hand, Instructors are specialists in such domains, but they are not supposed to have any special Computer Science or Programming skills. *Educational Modeling Languages (EML)* can help in

making this collaboration more efficient and agile [14]. Indeed, these EMLs provide instructors with a notation to describe their teaching methods. Therefore, in an idealized development setting, instructors might use suitable EMLs to formalize all the aspects of the learning process, while technicians might provide the hardware and software e-learning infrastructures required for the automatic deployment and execution of the formalized learning designs in standardized LMS.

Unfortunately, real-world EMLs must face a flexibility-usability gap, which seriously hinders the idealized scenario: the more expressive and flexible a EML is, the more difficult its use by instructors, non-experts in computer science. A good example is IMS Learning Design (IMS-LD), a powerful and expressive language, which is becoming a *de facto* encoding and interchange standard for learning methods [10]. Expressivity in IMS-LD is a necessity, since it must serve as a standardized interchange format for learning designs among heterogeneous LMS. However, IMS-LD advanced features, and, in particular, Level B's *condition system*, which support user adaptability, are difficult to use even for those proficient in computer science [2]. Indeed, condition systems are a kind of monolithic rule-based systems, such as those used in artificial intelligence, and they suffer from many of the production and maintenance issues identified in these systems [12]. Still, these features are essential to modeling adaptive learning flows, enabling us to automatically adapt LMS behavior to the individual needs of each particular learner [17]. On the other hand, more usable notations (e.g. the visual notation integrated in the LAMS authoring tool [5]) can either lack the required flexibility, or they must resolve difficult importation / exportation concerns in order to interoperate with the *de-facto* IMS-LD standard.

The main aim of our research efforts is to address the aforementioned flexibility-usability gap. For this purpose, we have defined a visual notation which preserves the structural aspects of IMS-LD (i.e. the level A), but which replaces the flexible but

cumbersome Level B's condition system with a more usable flow-oriented style of expressing activity sequencing [16]. We have integrated this notation in e-LD [15], an authoring tool for IMS-LD *Units of Learning* (the IMS term used for the courses) that we are developing at the Complutense University of Madrid (Spain). This tool integrates a semiautomatic, computer-aided importation system and an automatic exportation system. In this paper, we focus on exportation, that is, how to translate an e-LD representation, where sequencing is described using flowchart-like notations, into an IMS-LD representation. The problem itself can be meaningfully reduced to analyzing how to translate flow-oriented activity sequencing specifications to rule-oriented ones, and actually, that is the approach used in this paper. Moreover, to focus on the essential aspects of the translation and, at the same time, to make this paper self-contained we use compact flow-oriented and rule-oriented sequencing notations. These simplified notations preserve the essence of the original e-LD and IMS-LD paradigms.

The rest of the paper runs as follows: Section 2 presents a simple case-study, which models an adaptive learning flow, and which will be used to illustrate our approach. Section 3 presents the flow-oriented and rule-oriented notations chosen. Section 4 details and exemplifies the translation process itself. Section 5 describes some related work. Finally, section 6 presents some conclusions and lines of future work.

2. Case Study

In order to illustrate the different aspects presented in this paper we will use a specification of the conditional / adaptive sequencing of a set of learning activities.

This example models the sequencing of activities in a course on *Enology* (i.e. the science of wine). In this course, the learner must first attend a course introduction, then revise a book on enology written by the reputed expert *Mary Bebo* (all the names and the references in the example are fictitious), and finally take a test. After the test, the learning flow splits into different learning paths depending on the results. If the grade is low (< 5) the learner must review the introductory book written by *Joseph Birra* (another expert on the subject), and then retake the test. For acceptable scores (≥ 5 and < 8), the learner is compelled to visit the website "the art of wine tasting". Outstanding learners are compelled to participate in a workshop on the subject. Finally, the learner must prepare an essay on the course's topic. The flowchart of Fig. 1 sketches this case-study.

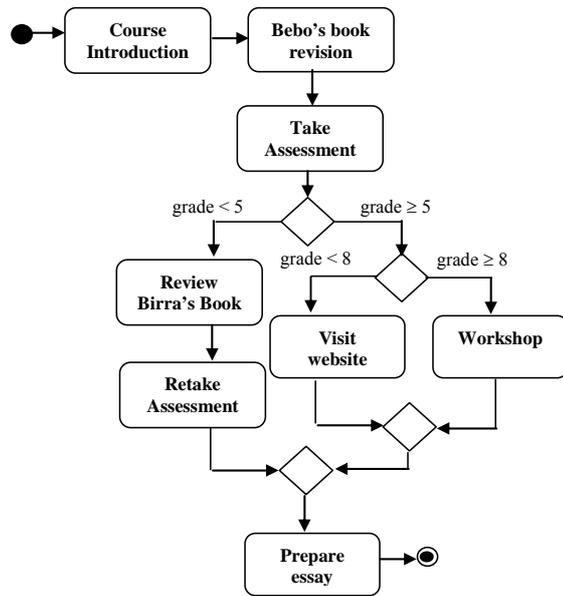


Fig. 1. Flowchart for the case-study

3. Notations and Sequencing Styles

The notation used in the case study promotes a *flow-oriented* style for sequencing activities. Rounded boxes are activities that must be performed during the learning process (e.g. *Course Introduction* or *Visit website*)¹. Arrows denote transitions between activities: when an activity finishes, the next activity is initiated (e.g. when the *Course Introduction* activity is completed, the learner must initiate the *Bebo's book revision* activity). Diamonds delimitate conditional learning flows, where the learner is compelled to follow one or another learning path depending on the result of checking a condition (e.g. if the grade is < 5 , the learner has to continue by reviewing Birra's book, or otherwise will continue with more advanced activities). Conditions are formulated on the values of a set of *properties*. Properties capture activity outcomes because property values are established by these activities (e.g. *grade* is a property that is set by the *Take Assessment* activity and whose value is a numeric value that ranges from 0 to 10). Finally, the filled circle initiates the learning flow, while the double circle finishes it. This graphical notation, which

¹ These activities can be further decomposed in the structural part of the design, as is the case with e-LD and IMS-LD, and they also have roles, learning objects and services assigned, as also occurs in the tool and language cited. However, these aspects are not relevant to the present discussion, which is exclusively focused on sequencing activities. See [16] for more information about how those complementary aspects can also be graphically described.

is a subset of the e-LD graphical notation relevant for activity sequencing, will be more formally described in section 4.3.

```

Rules      ::= Rules Rule
Rules      ::= Rule
Rule       ::= Exp → Actions .
Actions    ::= Actions , Action
Action     ::= Action
Action     ::= hide(activity-id) |
              show(activity-id) |
              property-id := Exp
Exp        ::= SExp Comp-op SExp | SExp
SExp      ::= SExp Add-op Term | Term
Term      ::= Term Mul-op Fact | Fact
Fact      ::= Un-op Fact | Atom
Atom      ::= number | true | false |
              defined(property-id) |
              completed(activity-id) |
              property-id | (Exp)
Comp-op    ::= < | > | ≤ | ≥ | = | ≠
Add-op     ::= + | - | or
Mul-op     ::= * | / | and
Un-op      ::= - | not

```

Fig. 2. Grammar for the rule-oriented notation.

As said before, IMS-LD uses a more expressive and flexible, although less usable, rule-based mechanism (*conditions*) for accomplishing conditional and adaptive activity sequences. Since introducing the full IMS-LD condition system is beyond the scope of this paper, we introduce a significant subset, which is sufficient to express all the designs describable with the previous flow-oriented notation, and which will let us describe the translation's essentials. We also use a more concise and compact textual notation instead of the XML binding proposed by the specification. This notation is established by the grammar of Fig. 2. According to this grammar, rules are of the form *condition* → *actions*.

The condition is an arbitrary numeric or boolean expression, of the same type as that used to label the diamonds' alternatives in the flow-oriented notation. As usual, non-zero values are considered *true*, while zero values are taken as *false*. It is also possible to ask if a property has a value (*defined* construct), or an activity is completed (*completed* construct).

Regarding actions, they are of three types: *show* an activity (the activity becomes visible for the learner), *hide* an activity (the activity becomes invisible), and *set* a property to a value. This last type of action is very relevant from a technical point of view since it turns the language into a computationally (Turing)-complete one [4], and therefore lets the notation embed a great variety of sequencing styles.

The operational semantics of the language is also straightforward: each time a change in a property is reported, the conditions of the rules are evaluated. If a rule's condition becomes true, the corresponding action

is executed. Sequencing is achieved by showing and hiding activities. As with the flow-oriented language, activities can also change the values of the properties.

Fig. 3 shows a little example of using the rule based

```

not completed(Course-Introduction) →
    show(Course-introduction).
completed-Course-Introduction and
not completed(Bebo's-Book-Revision) →
    hide(Course-introduction),
    show(Bebo's-Book-Revision).
completed(Bebo's-Book-Revision) and
not completed(Take-test) →
    hide(Bebo's-Book-Revision),
    show(Take-test).

```

Fig. 3. An example of rule-based activity sequencing specification.

notation for doing activity sequencing. The first rule shows the *Course Introduction* activity if it has not already been completed. When this activity has been completed, the activity *Bebo's book revision* is showed, provided that it has not been completed (rule 2). The third rule applies a similar strategy to continue with the *Take assessment* activity. This brief example reveals that, although the notation is very flexible, it is also difficult to use due to the atomicity of the steps and its reactive nature, which compels the instructors to be aware of the potential interactions and side effects between different steps in terms of the underlying global state. The next section addresses this issue, showing how the notation can support more specific sequencing mechanisms, such as the flow-oriented one, by means of automatic translations into the underlying rule-based substrate.

4. Translation

In order to translate flow-oriented specifications into rule-based ones we first need to figure out what a flow-oriented execution engine is. Then we need to devise how to encode the engine's behavior using rules. Finally, by using standard techniques in the translation of computer languages [1], we can design and implement an appropriate translator. Next points detail these aspects.

4.1. Flow-oriented execution engine

The engine for executing flow-oriented specifications can operate by being driven by a sequence of basic flow-oriented instructions. In every moment, an *instruction pointer* refers to the instruction to be executed. The execution starts with the first instruction. Then the instruction changes the engine's state and determines the next instruction to execute. The state itself is represented by a *memory of properties*, which stores the values of the properties

established by the activities. In Fig. 4 we summarize an engine’s instruction set which is sufficient for supporting our flow-oriented language, along with the intended meaning of these instructions.

Notice that the translation of flow-oriented specifications to the basic instruction set is similar to

Instruction	Intended Meaning
$expose(a)$	Exposes the activity a and waits until it is completed. Then updates the memory of properties, and continues with the next instruction
$jmpf(e, i)$	Evaluates the expression e . If the result is <i>false</i> , it jumps to the instruction i . Otherwise it continues with the next instruction
$jmp(i)$	It jumps to the instruction i .

Fig. 4. High-level architecture of the flow-oriented execution engine

the translation of a high-level structured program to assembly code. Therefore, this translation can be done using standard techniques in compiler construction [1]. However, our target language and machine is not this flow-oriented engine, but the rule-oriented one. Therefore, to fully undertake the translation, we first need to map the flow-oriented engine to the rule-oriented one.

4.2. Mapping the flow-oriented execution engine into a rule-based one

The rule-based execution engine also contains a memory of properties, which is updated each time an activity is completed. The main difference is the reactive execution behavior sketched in section 3. Thus, the instruction pointer does not exist, as rules are reactively executed when a change is registered in the property memory. Fortunately, the instruction pointer can be readily recovered by using a distinguished property (e.g. \$IP). This property can be initialized to 1. On the other hand, each instruction can be mapped into a set of rules (see Fig. 5):

- *Expose* instructions are translated into two rules. The first one makes the activity visible. The second one waits until it is completed, then hides the activity, and increments the \$IP property.
- Conditional jumping instructions are also translated into two rules: the first one checks that the jumping expression is true, and then it sets the \$IP property accordingly. The second one acts when the expression is false, incrementing the \$IP, which will refer to the next instruction.
- Finally, unconditional jumps are translated into a single rule, which fixes \$IP to the instruction’s argument.

Also notice how the rules’ conditions must check the \$IP value in order to guarantee the execution of the appropriate instruction.

Position	Instruction	Translation
i	$expose(a)$	$\$IP=i$ and not completed(a) \rightarrow show(a). $\$IP=i$ and completed(a) \rightarrow hide(a), $\$IP:=i+1$.
i	$jmpf(e, j)$	$\$IP=i$ and not(e) \rightarrow $\$IP := j$. $\$IP=i$ and (e) \rightarrow $\$IP := i+1$.
i	$jmp(j)$	$\$IP=i \rightarrow \$IP := j$.

Fig. 5. Mapping instructions into rules.

4.3. Translation Schemes

Finally, this section introduces the *translation schemes* of the flow-oriented notation into the rule-oriented one while taking into account the compiling techniques described in section 4.1 in combination with the mapping between the two engines presented in section 4.2 (for more details regarding syntax-directed definitions / translation schemes, see [1]). In Fig. 6 the first column formalizes the flow-oriented notation using a *visual grammar* (see [13] for more details on visual languages and visual grammars). The second column specifies the sequence of actions to be performed in the context of each syntax rule in order to carry out the translation. These actions can be either *primitive* or *non-primitive*. Non-primitive actions involve the translation into constituent structures (we denote this last one by enclosing the structure between brackets). Notice that the translation uses a global variable (ic), which counts the number of basic flow-oriented instructions considered during translation. Notice also that those basic instructions are never generated. Instead, we use *rule generation* procedures which generate the corresponding rules. For each instruction type it there is a generation procedure named *generate-it-rules*. This procedure takes the expected instruction’s position as a first argument and the expected arguments of the instruction itself as the others. With this information it generates the rules. Since these procedures follow the patterns presented in section 4.2, we will omit their detailed description. Besides, notice that there is considerable freedom while translating composite constructs (i.e. diamonds), since the patterns ensure that the order of the rules does not alter their global behavior. Indeed, the rules for skipping the *if* part when the first condition is false are generated once the *if* part has been generated. It avoids tedious *backpatching* strategies [1], and therefore facilitates the translation.

Taking this specification as a guide, the implementation of a translator is a direct task, as we

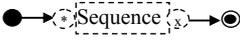
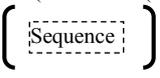
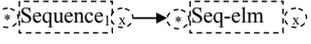
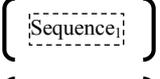
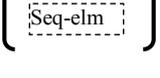
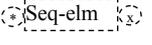
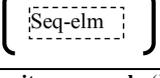
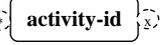
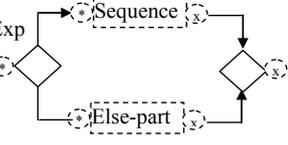
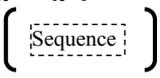
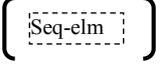
Syntax Rule	Sequence of Translation Actions
Flow ::= 	global ic ← 1 emit ("not defined(\$IP) → \$IP:=1.") 
Sequence ₀ ::= 	 
Sequence ::= 	
Seq-elm ::= 	emit-expose-rules (ic, activity-id.image) ic ← ic+1
Seq-elm ::= 	i-if ← ic ic ← ic+1  ic-else ←  emit-jmpf-rules (i-if, Exp.image, ic-else)
Else-Part ::= 	i-else ← ic ic ← ic+2  emit-jmp-rules (i-else, ic) emit-jmpf-rules (i-else+1, Exp.image, ic) return i-else+1
Else-Part ::= 	return ic

Fig. 6. Translation schemes for translating flow-oriented into rule-oriented activity sequencing descriptions.

have realized in the development of our e-LD tool. The result of applying such a translator to the flowchart of our case-study is partially shown in Fig. 7. Therefore, the complexity of rule-based specifications is hidden by the translator without sacrificing the benefits posed by standard representations as IMS-LD. In addition, other alternative sequencing styles (e.g. IMS Simple Sequencing [9]) can be also supported.

5. Related Work

Flow-oriented notations for describing learning designs have a long tradition in e-learning. In [7], this kind of notations is proposed for scripting interactive and highly-adaptive tutoring systems. In the IMS-LD specification itself, UML-based flow-oriented notations are proposed [8]. In [11] UML itself is adopted as an educational modeling language. LAMS, an authoring tool for learning designs, also adopts a flow-oriented-like authoring metaphor [5].

Regarding the translation of flow-oriented into rule-based notations, it is partially addressed in the IMS-LD not defined(\$IP) → \$IP :=1.

```

....
$IP=10 and not completed(Visit-website) →
  show(Visit-website).
$IP=10 and completed(Visit-website) →
  hide(Visit-website), $IP:=11.
$IP=9 and not(grade < 8) → $IP:=12.
$IP=9 and (grade < 8) → $IP:=10.
$IP=13 and not completed(Workshop) →
  Show(Workshop).
$IP=13 and completed(Workshop) →
  hide(Workshop), $IP=14.
$IP=11 → $IP:= 14
$IP=12 and not(grade ≥ 8) → $IP:= 14
$IP=12 and (grade ≥ 8) → $IP:=13

```

Fig. 7. Fragment of the rules that result of translating the case-study's flowchart.

Specification, where some guidelines are described for their subsequent IMS-LD encoding [8]. LAMS partially gives support for exporting to IMS-LD,

although until now complete support for IMS Learning Design Levels B (where the rule-based condition system arises) has not been reported. In [6], transformational techniques in model-driven engineering are proposed as a mechanism to map flow-oriented patterns into IMS-LD (and also to recognize these patterns in existing IMS-LD Units of Learning). However, none of these proposals reports full automatic support for the translation process.

6. Conclusions and Future Work

This paper presents an approach for the automatic translation of e-learning flow-oriented activity sequencing notations into a rule-based one, such as that promoted by the IMS-LD level B condition system. This approach makes a meta-linguistic use of the rule-based substrate, by mapping the computation model of the flow-oriented engine onto it. As a result, instructors can author their learning designs in a more suitable and user-friendly notation. The automatic translation support makes it possible to deploy these designs in rule-based platforms in order to address the standard interoperability issues (i.e. automatic execution of the design in different LMS). We have implemented the approach in our e-LD authoring system, which is able to make a complete exportation of flow-oriented graphical designs to IMS-LD.

Currently we are working on reducing the number of rules generated by the translation and also on supporting other notations and sequencing styles and, in particular the IMS Simple Sequencing language [9] that is used in the SCORM reference model.

Acknowledgements

The Spanish Committee of Science and Technology (projects TIN2005-08788-C04-01, Flexo-TSI-020301-2008-19 and TIN2007-68125-C02-01) has partially supported this work, as well as the Complutense University of Madrid (research group 921340, Santander/UCM Project PR34/07 – 15865) and the EU Alfa project CID (II-0511-A).

References

- [1] Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D (2007). *Compilers: principles, techniques and tools* (2nd edition). Addison-Wesley.
- [2] Burgos, D., Moreno-Ger, P., Sierra, J.L., Fernández-Manjón, B., Koper, R.(2007). *Authoring Game-Based Adaptive Units of Learning with IMS Learning Design and <e-Adventure>*. *International Journal of Learning Technology*, 3(3), 252-268. 2007.
- [3] Caeiro, M. Marcelino, M.J., Llamas, M., Anido-Rifón, L., Mendes, A.J. (2007). *Supporting the Modeling of Flexible Educational Units PoEML: A Separation of Concerns Approach*. *Journal of Universal Computer Science* 13(7):980-990.
- [4] Cutland, N.J (1980). *Computability: An Introduction to Recursive Function Theory*. Cambridge Univ. Press
- [5] Dalziel, J. (2006), *Lessons from LAMS for IMS Learning Design*. Sixth IEEE Int. Conf. on Advanced Learning Technologies (ICALT'06), 2006.: 1101-1102.
- [6] Dodero, J.M., Tattersall, C., Burgos, D., Koper, R. (2007) *Transformational Techniques for Model-Driven Authoring of Learning Designs*. 6th Int. Conference in Web Based Learning (ICWL 2007): 230-241.
- [7] Ibrahim, B.,1989. *Software Engineering Techniques for CAL*. *Education & Computers* 5, 215-222
- [8] IMS (2003). *IMS Learning Design Information Model - Version 1.0 Final Specification*. Retrieved November 6, 2008 from: http://www.imsglobal.org/learningdesign/ldv1p0/imsld_infov1p0.html
- [9] IMS (2003). *IMS Simple Sequencing Information and Behavior Model - Version 1.0 Final Specification*. Retrieved November 6, 2008 from: http://www.imsglobal.org/simplesequencing/ssv1p0/ims_ss_infov1p0.htm
- [10] Koper, R. and C. Tattersall (2005). *Learning Design - A Handbook on Modelling and Delivering Networked Education and Training*. Heidelberg, Springer Verlag.
- [11] Laforcade, P (2005). *Towards a UML-based educational modeling language*. Fifth IEEE Int. Conf. on Advanced Learning Technologies, (ICALT'05), 2005. pp 855-859.
- [12] Li, X. (1991). *What's So Bad About Rule-Based Programming?* *IEEE Software* 8: 103, 105.
- [13] Marriott,K., Meyer,B., Wittenburg,K.B.A (1999). *Survey of Visual Language Specification and Recognition*, in Marriot,K., Meyer,B (eds). *Visual Language Theory*, pp. 5-85, Springer-Verlag.
- [14] Martínez-Ortiz, I., Moreno-Ger, P., Sierra, J.L., Fernández-Manjón, B (2007). *Educational Modeling Languages: A Conceptual Introduction and a High-Level Classification*, in Fernández-Manjón et al. (eds) *Computers and Education: E-learning - from theory to practice*. Springer-Verlag New York, Inc.: 27-4
- [15] Martínez-Ortiz, I., Moreno-Ger, P., Sierra, J.L., Fernández-Manjón, B. (2007). *Supporting the Authoring and Operationalization of Educational Modelling Languages*. *Journal of Universal Computer Science* 13(7): 938-947.
- [16] Martínez-Ortiz, I., Moreno-Ger, P., Sierra, J.L., Fernández-Manjón, B (2008). *A Flow-Oriented Visual Language for Learning Designs*. 7th International Conference on Web-based Learning (ICWL2008). Jinhua (China), LNCS 5145: 486-496.
- [17] Specht, M., Burgos, D. (2007). *Modeling Adaptive Educational Methods with IMS Learning Design*. *Journal of Interactive Media in Education* 2007/08. Retrieved November 6, 2008 from: jime.open.ac.uk/2007/08.