2010 10th IEEE International Conference on Advanced Learning Technologies

Managing the Production and Evolution of e-learning Tools with Attribute Grammars

Bryan Temprado-Battad, Antonio Sarasa-Cabezuelo, José-Luis Sierra
Dpto. Ingeniería del Software e Inteligencia Artificial. Fac. Informática. Universidad Complutense de Madrid. 28040 Madrid (Spain)
{bryan,asarasa,jlsierra}@fdi.ucm.es

Abstract: **Many e-learning tools are based on domain-specific languages (DSLs) targeted to the educational domain. Thus, methods and techniques from the programming language community can help in developing these tools. In this paper, we show how attribute grammars, a well-known declarative specification method for the syntax and semantics of programming languages, can facilitate the production and subsequent evolution of e-learning tools. We also describe how we produced and extended <e-Tutor>, a courseware system supporting an XML-based DSL, by using XLOP (XML Language-Oriented Processing), a meta-tool supporting attribute grammars for the development of XML processing applications.**

Keywords: ***e-learning tools, attribute grammars, domain-specific languages, XML processing, tutoring systems***

I. INTRODUCTION

E-learning systems and tools usually rely on educationally-targeted domain-specific languages (DSLs), which describe different aspects of an e-learning system (such as assessments, sequencing of learning activities, or the structure and metadata of learning objects). IMS specifications are an example [3], but specific applications often have their own DSLs as well [9].

Although e-learning DSLs are simpler than general-purpose programming languages [5], their conception, specification and processing remains a difficult task that demands specialized and systematic approaches. Fortunately, computer scientists have developed an arsenal of tools for managing the complexity of producing and extending a programming language [5]. An especially relevant technique is the formalism of *attribute grammars*, conceived by Donald E. Knuth during the 1960s [4][6].

Attribute grammars are attractive because they neatly balance syntax-directed declarative specification with efficient implementation. On the one hand, attribute grammars allow high-level and declarative specification of an educational DSL, in a completely syntax-directed way. Attribute grammars also make it easy to change the specification when the DSL changes (for instance, to add a new feature). On the other hand, meta-tools can process attribute grammars automatically to produce efficient implementations of language processors (such as interpreters and translators). In this paper, we show how developers can take advantage of these features.

The rest of the paper is organized as follows: Section II gives a brief introduction to attribute grammars. Section III describes how attribute grammars can be used during the production and evolution of e-learning tools. We give a case study in Section IV, which describes how the production and evolution of *<e-Tutor>*, a tool for building Socratic tutorials [2], was managed with the help of the attribute grammar-based meta-tool XLOP (*XML* Language-Oriented Processing) [7][8]. Finally, Section V presents some conclusions and ideas for future work.

II. ATTRIBUTE GRAMMARS

An attribute grammar specifies how to process the sentences of a language. Attribute grammars are *syntax-directed*, which means that the processing of a sentence is dictated by its syntactic structure [1].

A. Describing structure with context-free grammars

In order to describe the structure of the sentences of a language (that is, the structural syntax of this language), an attribute grammar include a *context-free grammar* as its basic skeleton. A context-free grammar uses *syntax rules* or *productions* to define the syntax of its sentences. Syntax rules are rules of the form $A \rightarrow \alpha$, where A is a *non-terminal* symbol, and α is a (possibly empty) sequence of *non-terminal* and *terminal* symbols. Terminal symbols represent atomic constructions, while non-terminal symbols represent composite constructions.

Context-free grammars impose a tree-shaped structure on each sentence of the language; this structure is called the *parse tree* of the sentence. In a parse tree, internal nodes are labeled with non-terminal symbols, leaf nodes are labeled with terminal symbols (or λ, the *empty* string), and parent-child relationships are dictated by syntax rules.

B. Describing processing with semantic attributes and semantic equations

Attribute grammars introduce a processing model based on the association of semantic attributes with the symbols of parse trees, and on the computation of semantic values for such symbols. This computation is known as *semantic evaluation*. There are two kinds of attributes: *synthesized* and *inherited*. The value of a synthesized attribute represents the *meaning* of the symbol, while the value of an inherited attribute provides additional *contextual information* needed during semantic evaluation. The synthesized attributes of

978-0-7695-4055-9/10 $26.00 © 2010 IEEE
DOI 10.1109/ICALT.2010.124

terminal symbols are called *lexical attributes*, and they must be set during lexical analysis.

To describe how to compute semantic values, syntax rules are enriched with *semantic equations*. There is a semantic equation for each synthesized attribute in the rule's left-hand side, and another one for each inherited attribute in the rule's right-hand side. Semantic equations dictate how to compute values of attributes with *semantic expressions*. The most basic semantic expressions are references to other attributes in the production. *Semantic functions* form more complex expressions from simpler expressions.

Semantic evaluation must respect *dependencies* between semantic attributes. An attribute depends on those attributes that are required to compute it. Therefore, before computing the value of an attribute, the values of the attributes on which it depends must have been computed already. Aside from this basic constraint, evaluation order does not matter. This insensitivity to evaluation order engenders a highly modular *dependency-driven* computation style, facilitating the evolution of language processors. Each evolution step usually adds new rules, semantic attributes and semantic equations. The dependency-driven computation style restructures semantic evaluation automatically. It also means attribute grammars are higher-level than other syntax-directed processing styles (for example, *translation schemes*: context-free grammars with interleaved semantic actions that fire during parsing [1]).

(a)

```
Activity → Activity and then Another-Activity
    Activity0.seq = mkAndActivity(Activity1.seq,
                                  Another-Activity.seq)
Activity → Activity  or Another-Activity
    Activity0.seq = mkOrActivity(Activity1.seq,
                                 Another-Activity.seq)
Activity → Another-Activity
    Activity.seq = Another-Activity.seq
Another-Activity → b-act
    Another-Activity.seq = mkAct(b-act.lex)
Another-Activity → (Activity)
    Another-Activity.seq = Activity.seq
```

(b)

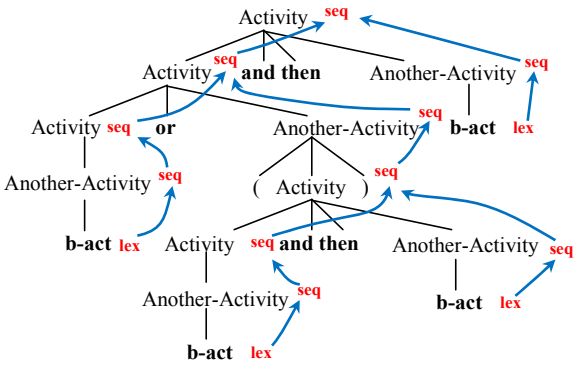

Figure 1. (a) Attribute grammar to translate descriptions of learning sequences into data structures; (b) a parse tree and the dependencies between the attributes of its symbols.

In Figure 1a, we show an attribute grammar for a simplified activity sequencing language; the grammar translates learning sequences expressed in the source language to data structures suitable for playing them. $X.a$ refers to an attribute a of X in a production. A sub-index, attached to the grammar symbol, disambiguates; thus X_i represents the ith occurrence of the grammar symbol X in the syntax rule. The attribute *lex* of the terminal symbol **b-act** is a lexical attribute containing the actual activity name. The synthesized attribute *seq* contains the resulting data structures. Figure 1b shows an example of dependencies between the attributes of a parse tree.

III. The Role of Attribute-Grammars in the Development of e-Learning Tools

This section describes a process model for the development of e-learning tools using attribute grammars. Subsection III.A outlines the model itself. Subsection III.B discusses its different phases.

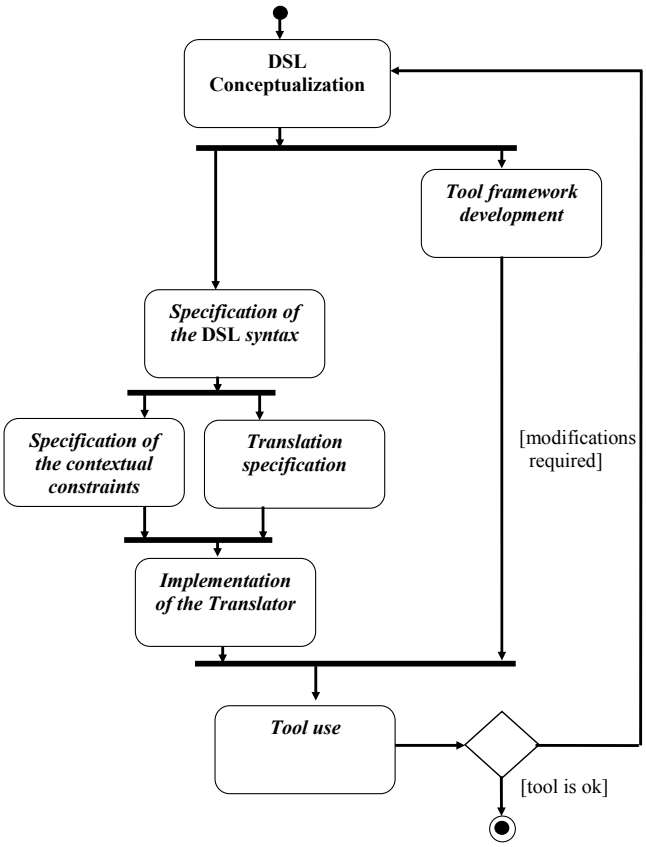

Figure 2. Process model for managing the production and evolution of e-learning tools with attribute grammars.

A. The process model

Figure 2 outlines the development process model. According to this model, the development of an e-learning tool is intrinsically language-driven. Development begins by specifying the DSL that underlies the tool. Then, it proceeds with the construction of the tool itself. This construction is clearly split into two different parts: a *translator* and a *tool framework*. The relationship between these two components

is that the translator translates descriptions in the DSL into instantiations of the tool framework. This explicit separation dramatically improves tool production and evolution. On the one hand, the translator and the tool framework can be developed, improved and enhanced independently, provided that the basic interfaces are preserved. On the other hand, the translator can be developed using specialized techniques (attribute grammars in particular). Also, notice that tool evolution is especially relevant in this context, since as Figure 2 makes clear, the process model is incremental and iterative in nature: when the tool users (both instructors and students) discover a requirement not met by the current version of the tool, developers must add this functionality. In this case, both the tool framework and the translator must change. For this purpose, and as indicated in the previous section, the declarative nature of attribute grammars can be extremely helpful.

B. *Development phases*

Development begins with the DSL *conceptualization* phase. This phase amounts to analyzing the tool's requirements and distilling them into the educational DSL. Both computer scientists and experts in educational and instructional design must participate in this activity. Collaboration between these two kinds of experts is required in order to ensure an adequate trade-off between pedagogical and purely technological requirements in the DSL.

The next two phases are *specification of the* DSL *syntax*, and *specification of contextual constraints*. A contextual constraint is an additional structural condition to be satisfied by the DSL sentences, which is not captured by the underlying context-free grammar. Such constraints are expressed using semantic attributes and equations that specify whether constraints are preserved or violated. *Translation specification* means describing how programs are translated to instantiations of the tool framework using additional semantic attributes and equations. The last phase in this branch is the *implementation of the translator*. Developers can implement the translator manually with the techniques described in standard compiler textbooks [1], or a suitable *meta-tool* can be used instead. A meta-tool can produce running translators from the attribute grammar-based specifications provided during the previous specification phases. Meta-tools are essential components in this language-driven development process.

The *tool framework development* phase provides the tool's conventional computational machinery (as a tool-specific object-oriented framework). Finally, during the *tool use* phase, domain experts (such as instructors, researchers, and students) use the tool in their work. Through their regular use of the tool, users may discover new desired functionalities or infelicities. Developers can take these discoveries into account when extending the tool.

IV. A CASE-STUDY

In order to illustrate the development process model outlined in the previous section, we will focus on the production and further development of *<e-Tutor>*, a tool for building Socratic tutorials based on the work of Alfred Bork and his colleagues during the 1980s [2]. Subsection IV.A describes the initial conception and initial versions of *<e-Tutor>*. Subsection IV.B describes successive iterations.

A. *Initial conception and initial versions*

Our main motivation for formulating *<e-Tutor>* was to experiment with language-driven techniques, markup-based DSLs in particular, in the development of e-learning tools. For this purpose, in the first incarnations of the DSL conceptualization phase, we conceived an XML-based language for documents describing Socratic tutorials. Figure 3 shows an excerpt from the DTD for this language, which establishes the markup for the description of the *problems* in the tutorial documents: problems start with (a possibly empty sequence of) *tutorial elements*, followed by a *question point*, in which the learner is interrogated. In the initial versions, we only conceived *text* and *images* as tutorial elements (that is, as the chunks of information shown to the learner), as well as text-based interaction (the learner should enter a string as response to the questions). We also designed a rudimentary user interface (see Figure 4).

```
...
<!ENTITY % tElement "(Text|Image)">
<!ENTITY % qPoint "(QuestionPoint)">
...
<!ELEMENT Problem (%tElement;*, %qPoint;)>
...
```

Figure 3. Excerpt from the <e-Tutor> DTD

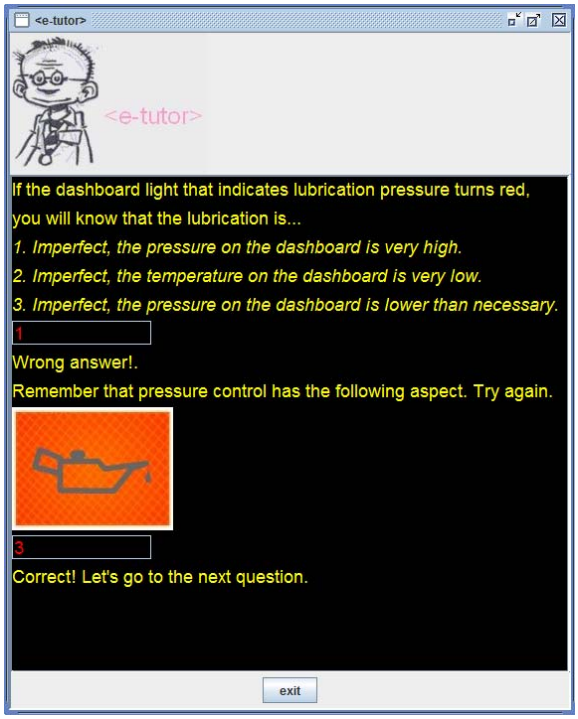

Figure 4. Initial user interface

Originally, we developed the software in an *ad hoc* manner, by hand-coding the translator directly in Java (with the support of a standard DOM-based XML processing API). However, when the language became more complex, this undisciplined development method became unsustainable. This problem encouraged us to formalize the language and the translation using attribute grammars, in the specification phases detailed in the previous section. Then, we considered using meta-tools to automate the generation of the translator. Our first solution was based in a generative framework embedded in Java [10], which we modified incrementally, eventually yielding aforementioned XLOP meta-tool.

```
...
Problems ::= Problems Problem {
  unlinkedTutorial of Problems(0) =
    addNewProblem(unlinkedTutorial of Problems(1),
                  id of Problem, elem of Problem)
  unlinkedTutorial_i of Problems(1) =
                  unlinkedTutorial_i of Problems(0)
  unlinkedTutorial_i of Problem =
                  finalTutorial of Problems(1)
  finalTutorial of Problems(0) =
                  finalTutorial of Problem
}
...
```

Figure 5. Excerpt from the initial XLOP specification for the <e-Tutor> translator.

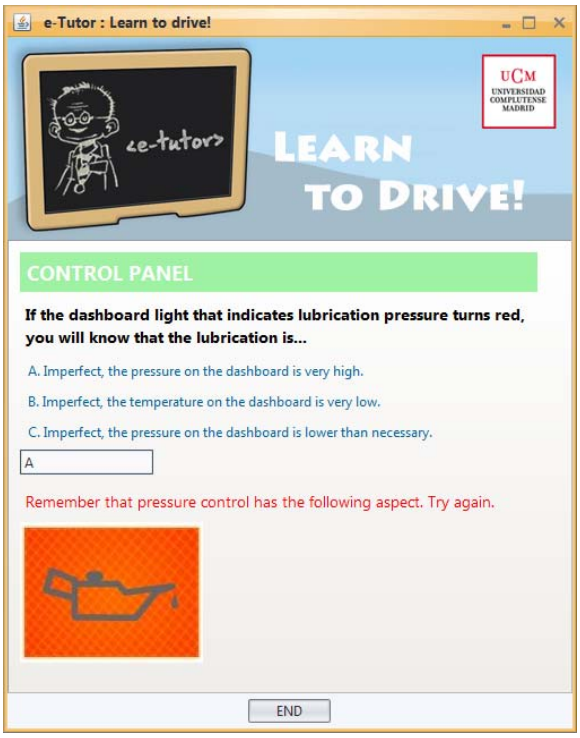

Figure 6. Enhanced user interface

XLOP fully supports attribute grammars as an independent specification language. This specification language is targeted to the syntax-directed aspect of the translator. Semantic functions must be implemented by auxiliary code (the methods of a Java *semantic class*). Therefore, XLOP divides XML processors into two well-defined layers: the *linguistic* layer, managed by the XLOP specification language, and the *application-specific logic* layer, managed by conventional software development methods. The semantic class connects these layers. Figure 5 shows an excerpt from the XLOP specification for the translation of XML documents into running tutorials in *<e-Tutor>*.

B. *Successive iterations*

Once we reached a stable version of *<e-Tutor>*, we implemented some improvements to the user interface and to the internal *reproduction* logic (see Figure 6). We made these changes without changing the XLOP specification, in an indirect consequence of the separation of concerns promoted by attribute grammars and XLOP. Indeed, semantic functions, and therefore the semantic class, constitute a clearly defined interface between the translator and the tool framework. It lets developers change the tool framework without further changes to the XLOP specification, provided that no new features have been added to the DSL.

```
...
Problems ::= Problems Problem {
   tutorialCreated_i of Problems(1) =
     tutorialCreated_i of Problems(0)
   tutorialCreated_i of Problem =
     tutorialCreated_i of Problems(0)
   problemsCreated of Problems(0) = after(
     problemsCreated of Problems(1),
     newProblem(id of Problem, elem of Problem))
}
...
```

Figure 7. Excerpt from the enhanced XLOP specification.

```
...
<!ENTITY % tElement "(Text|Image |
                      video | flash | youTube)">
<!ENTITY % qPoint "(QuestionPoint |
                      imageQuestionPoint)">
...
<!ELEMENT Problem (%tElement;*, %qPoint;)>
...
```

Figure 8. Extension of the <e-Tutor> language to support videos, Flash™ presentations, YouTube™ streaming video, and image based interaction (excerpt from the DTD)

The next step in the development of the tool was to refactor the XLOP specification *itself*. We realized that the instances of the semantic class, given that it was a conventional Java class, could maintain state in addition to implementing semantic functions as methods. Encapsulating state in instances allowed us to make the main structures to be used during translation instance fields in the semantic

class, and to refactor the XLOP specification to maintain dependencies between *state changes* instead of between *values*. This specification technique greatly simplified the original specification, avoiding several pairs of *inherited–synthesized* attributes whose sole purpose was to maintain state (see Figure 7). In this case, we were able to modernize the translator without touching a single line of the tool framework. Thus, we saw that the separation of concerns promoted by attribute grammars paid off again.

Finally, we extended the *<e-Tutor>* language to provide richer interaction mechanisms based on the selection of images, as well as richer tutorial elements: video, Flash™ presentations and streaming videos available on YouTube™ (see Figure 8 and Figure 9). Once we added the machinery for interpreting the new interaction mechanism and for rendering the new presentation media to the tool framework, extending the translator was straightforward and limited to the addition of a rule for the new interaction mechanism and for each new media type (Figure 10).

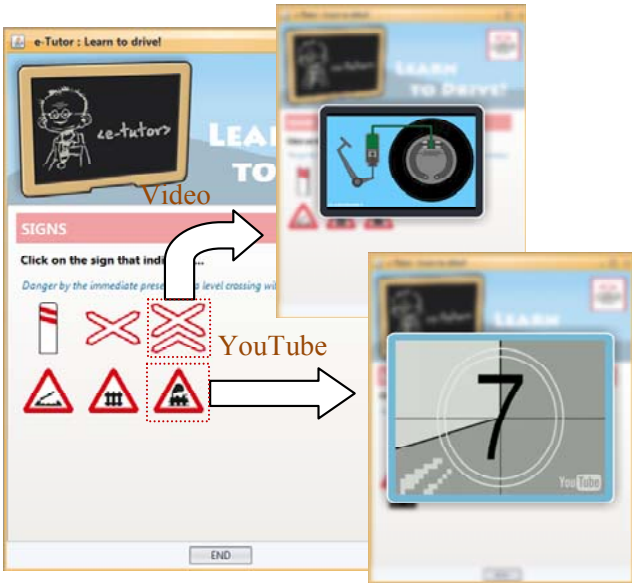

Figure 9. A tutorial that includes the features added in Figure 8.

```
...
Element ::= <YouTube/> {
  elem of Element = after(
   tutorialCreated_i of Element,
    newYouTube(path of <YouTube>,
                dialog of <YouTube>,
                newColor(dcolor of <YouTube>),
                continueBtn of <YouTube>,
                cleanScreen of <YouTube>))
}
...
```

Figure 10. Rule added to the XLOP specification regarding the support of youTube™ videos.

V. Conclusions and Future Work

In this paper, we have shown how attribute grammars, a well-known formalism used to specify programming languages and their processors, can be also used to manage the production and evolution of e-learning tools, when these tools are based on educational DSLs. Attribute grammars promote decomposing the tool into two well-separated components, the translator and the tool framework, which in many cases can be independently improved and extended. In addition, the declarative and modular nature of the specification formalism makes the translator easier to develop and maintain. This is especially true when a meta-tool is used. We have illustrated the approach with the production and extension of the experimental courseware system <e-Tutor>. We managed this process with XLOP, a meta-tool supporting attribute grammars to build XML processors.

Currently we are refining the XLOP meta-tool to better support modularity. We are also planning to apply the approach to existing e-learning specifications (in particular, players for IMS QTI and for IMS Simple Sequencing). Finally, we are working on extending XLOP to support a mixture of markup-based, textual and graphical syntax.

Acknowledgements

This work has been supported by the project grant Santander/UCM PR34/07-15865.

References

[1] Aho AV, Lam MS, Sethi R, Ullman JD. 2006. Compilers: principles, techniques and tools. Addison-Wesley; 2006

[2] Bork, A. 1985. Personal Computers for Education. New York, NY, USA: Harper & Row Publishers, Inc.

[3] Friesen N. 2005. Interoperability and Learning Objects: An Overview of e-Learning Standardization. Interdisciplinary Journal of Knowledge and Learning Objects. 1: 23-31

[4] Knuth, D. E. 1968. Semantics of Context-free Languages. Math. System Theory 2(2), 127–145. See also the correction published in Math. System Theory 5, 1, 95–96

[5] Mernik M, Heering J, Sloane AM. 2005. When and how to Develop Domain-Specific Languages. ACM Computing Surveys

[6] Paaki, J. Attribute Grammar Paradigms – A High-Level Methodology in Language Implementation. 1995. ACM Computing Surveys, 27(2): 196-255

[7] Sarasa, A., Sierra. J.L, Fernández-Valmayor, A. 2009. Processing Learning Objects with Attribute Grammars. 8th IEEE Int. Conf. in Advanced Learning Technologies

[8] Sarasa, A, Temprado-Battad, B, Sierra, J.L, Fernández-Valmayor, A. 2009. XML Language-Oriented Processing with XLOP. 5th Int. Symp. on Web and Mobile Information Services

[9] Sierra JL, Fernández-Valmayor A, Fernández-Manjón B. 2006. A Document-Oriented Paradigm for the Construction of Content-Intensive Applications. Computer Journal. 49(5): 562-584

[10] Sierra,J.L, Fernández-Valmayor,A., Fernández-Manjón, B. 2008. From Documents to Applications Using Markup Languages. IEEE Software 25(2), pp. 68-76