An application framework for E-learning

JAN HERMAN VERPOORTEN
Institute of Information and Computing Sciences
Utrecht University
Padualaan 14, 3584 CH Utrecht University
THE NETHERLANDS

Abstract: - The development of learning technology systems is expensive and time consuming. This applies to both the system and the educational content. This article describes an effective solution in the form of an application framework. The framework is capable to deliver highly interactive content and is suitable for all kinds of instruction, with or without adaptive behavior. It can be handled by educational designers and requires minor technical skills. The framework architecture is designed for change by programmers.

Key-words: - Architectures, XML, Software Design and Development, Educational Software.

1 Introduction

E-learning applications, following the IEEE recommendations [1] from now on called Learning Technology Systems (LTSs), are highly interactive applications, possibly with complex behavior. They also contain often huge amounts of educational content. The effect of both factors is that many projects face a high cost. Cost does matter because the current demand for LTSs is great, incomparable with other software applications. If LTSs become more incorporated in the daily practice, the call will be even greater. So, huge resources (people, time and money) are needed for development and maintenance.

An application framework (called *ClassMate*) specifically focused on LTSs has been developed. A framework can reduce development cost largely because both design and code can be reused over and over. Unfortunately the development of effective application frameworks is difficult. Effective reuse is only achieved by a not too complex structure and the capacity to meet the requirements in a broad application variety. ClassMate has an ancestor [2], deployed in a variety of domains and instructional levels. Application types range from laboratory programs for educational research [3,4] to computer-based training and testing [4,5]. Useful experiences were gathered and contributed largely to the current design. A browser-based prototype (ECMA script) of this design was constructed first. It was succesfully deployed in two multimedial web-based learning projects in order to validate the new design concepts. This article discusses the final result.

Unlike other application frameworks, ClassMate is more than an architecture for a component library. In a way, it is a self-assembling program. A developer creates a specification, written in an XML grammar. When the framework is launched (as an applet on a web page or as an application), it reads the specification and starts instantiating and assembling the specified components. So, developing an LTS is equivalent to writing a specification which enables very rapid application development, requires minor technical skills but assumes a sound knowledge of the framework modeling.

In the next section, this modeling is discussed. Section 3 decribes the modeling of educational content in ClassMate. Both issues are related to the XML grammar, needed for specification. Section 4 finalizes the article with a conclusion.

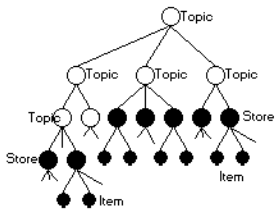
2 Modeling the application framework

The scope of an application, developed with the ClassMate framework is called a *session*. In a session specification three matters stand central: the content profile, the session course and controlling either element by the program, the learner or both. Any control causes runtime dynamics. An example of such dynamics is an adaptive test, presenting items whose difficulty is relative to the learner's progress. Content profile, session course and runtime dynamics will be discussed next.

2.1 Content profile

A content profile is expressed with three terms: topic, store and item. Topic is only a label for lesson content and is completely neutral as to application type or granularity. For example the topics 'nouns'

and 'verbs' in a language learning lesson. From the learner its perspective, a session is a series of individual pieces of lesson content, called items. In between topic and item stands store, which is an abstraction for storage. The relations between topic, store and item are expressed in a tree: content tree. Topic nodes contain either topic or store nodes; store nodes only contain items. Because a session specification does not contain physical content, an item node does not not contain the physical item but knows its storage location and meta data such as an ID or a style sheet. Figure 1 shows a possible content tree structure and the XML grammar of a topic node.




```
<element name='TopicSpecNode'>
  <complexType>
    <choice>
      <element ref='cm:TopicSpecNode'
        maxOccurs='unbounded'/>
      <element ref='cm:StoreNode' maxOccurs='unbounded'/>
    </choice>
    <attribute name='name' type='string' use='required'/>
    <attribute name='learnerControllable'
      type='boolean'
      use='optional' default='true'/>
    <attribute name='maxWeight'
      type='integer' use='optional'/>
    <attribute name='minWeight'
      type='integer' use='optional'/>
    <attribute name='nodeShufflerClassName'
      type='string' use='optional'
      default='DefaultNodeShuffler'/>
    <attribute name='nodeType'
      type='cm:NODETYPE' use='optional' default='OR'/>
    <attribute name='selected'
      type='boolean' use='optional' default='false'/>
    <attribute name='weight' type='integer' use='optional'/>
  </complexType>
</element>
```

Fig.1. Possible content tree structure and XML grammar of a topic node

As you can see in figure 1, a topic node has many attributes such as weight, minWeight, maxWeight, selected and nodeType. All about the same is the case for store nodes. The attributes specify the relative importance of a node and enable *dominance* of a node on to its children. This way, all kind of content profiles can be realised. For example each other including or excluding topics, stores or items can be expressed: If a store has the attribute-value pair nodeType="AND", selection of that store (or of one item inside) implies the selection of all items: the none_or_all dominance.

The relative importance in the content profile of a topic or store node also defines the number of items assigned to it during runtime. An algorithm calculates this number, also taking the various weights and nodeTypes into account. When during runtime content tree is asked for an item, a node shuffler (see figure 1) defines which one is returned: Each node uses its own shuffler object. The framework contains different shufflers but, as with most components, an external component, implementing a particular interface or abstract class, may be delivered.

As many other aspects of the framework, a content tree doesn't need to be static all the time. It may be changed by user control, program control or by both: This is the possible dynamic behavior of a session. I will further explore the meaning of dynamic behavior to a content tree in the next section.

2.2 Session course

The course of a session is controlled either by the learner, the program or both. A distinction is made between initial and runtime issues. The bootstrap component of the framework deals with all initial issues, such as reading in the specification. So, each session starts with a bootstrap: The specification defines exactly what must or can be done both by the program and the learner. This also influences the content tree, for example if a learner is allowed to make own choices. In the XML grammar of figure 1 this is represented by the learnerControllable and selected attributes. To clarify, please look at the hypothetical specification in figure 2. The learner's choices are reflected by thick lines. As you can see in the figure, topic c has nodeType='AND'. The effect is that, because the learner selected either c, c1 or c2, automatically the c node as well as its branches, here c1 and c2, were selected. The learner also choose node b2 or b3, without further consequences, although these selections were checked against the constraint on the root, which is a minWeight=300 and a maxWeight=1000. This time the learner selected a total sum of 450, so everything is correct. When the learner finishes the bootstrap, the tree removes its redundant branches. This way the final content tree is realized (see figure 2).

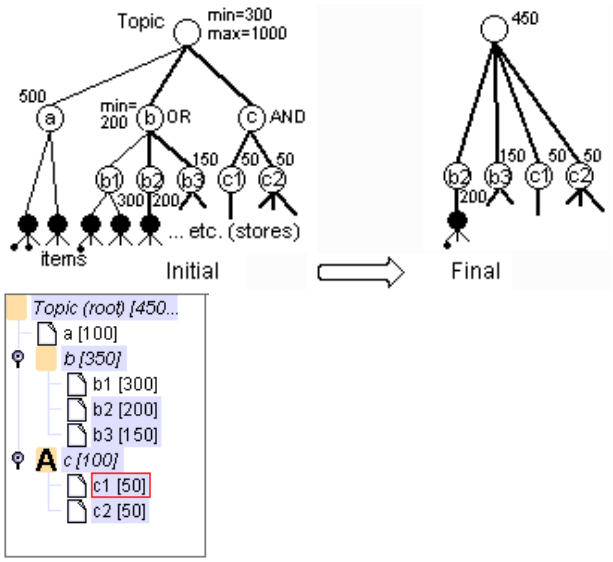


Fig.2. Initial and final content tree; screen fragment below

The runtime control issues of the session course differ greatly from the initial ones. The framework has a number of instances to define the behavior of a session. One such instance is the run strategy which bundles aspects like item mode (modal or modeless), iteration (the number of loops), its control (learner or program), menu and toolbar items and so on. This runtime control is (of course) also part of an specification. The learner part of it is mainly contained in the user interface specifications. This applies not only to menu or toolbar items. Also different tasklist managers and item managers can be defined, hiding either all aspects behind the scenes or giving the learner an influence on the session course.

Program control on the progress is achieved by rules in a specification. During the course of a run strategy, the learner's progress causes different events and session states. These can be referred to in rules. Each rule contains an expression and an action list. If for example the learner performs poorly on a topic, its relative weight in the content profile may be increased. A possible expression in an XML instance document for such a situation is showed in figure 3 below. In this fragment a run strategy contains two rules. They effect adaptive behavior: more items are assigned to topic 'c' if necessarily.

```
<RunStrategy ..... >
  .....
  <Rule value=' (c.getScore()<0.5) && (c.getItemsDone()>3)'
    actionList=' c.setWeight(100)' />
  <Rule value=' (c.getScore()<0.5) && (c.isImplemented())'
    actionList=' c.setWeight(200);segment.setItems(20)' />
</RunStrategy>
```

Fig.3. Defining dynamic behavior in a run strategy

3 Content modeling

In ClassMate, educational content is synonymous to items. An item is a modeling issue: it is the smallest independent content unit. Similar to the session specification, each item is a single specification in an XML grammar. When asked for an item, the content tree (part of the session specification) returns the item's metadata such as the location. The framework then reads the item specification and starts constructing the item by instantiating and assembling all specified components: Viewed from the application framework, an item is a container object for viewers, interactions, rules, inner items and a view tree. Finally the framework renders the item on the screen, as defined in the view tree part of the item specification. After the learner-system transactions the item manager on duty stores the item.

Viewers apply to non-interactive data. They may contain some interaction (e.g. a zoom function in an image viewer), but such interaction is strictly for program handling purposes and is not part of the instructional experience. Interactions perform the (educational) learner-system interactions. Transactional behavior is specified in rules and is largely performed by inner items: items, recursively contained (or specified) in items. Rules are not restricted to connecting learner events to inner items: everything in the entire framework can be manipulated, similar to the rules in the session specification as mentioned before. The viewer and interaction parts of an item specification will be discussed individually next.

3.1 Viewers

Viewers are modeled on three aspects: data model, visible granularity and the data type of the latter. Three basic data models are distinguished: singleton, list and tree. The framework contains a number of viewers, although all handle an intrinsic content type. For singletons, ClassMate uses the MIME (Multipurpose Internet Mail Extensions) type system. The framework is able to 'deliver' a suitable viewer if a specification contains a known MIME declaration. Custom MIME viewers can be registered by means of the session specification.

A singleton is a single entity, containing intrinsic content. A GIF bitmap image for example has a singleton data model and intrinsic content. Complexity arises when data contains intrinsic content and other data models. Now we have a

container at the root of a complete hierarchy (figure 4). Usually, such a container is called a document.

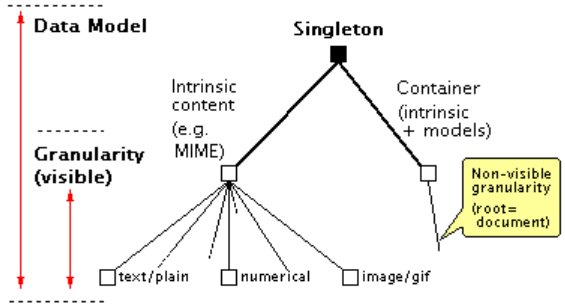


Fig.4. Data model, granularity and data type

The difference between singletons and container documents is made because of the difference in inner complexity. In case of a singleton with a visible granularity, a viewer can be specified by its MIME type. It is more difficult in contrast to specify a container viewer by a comparable and reliable generic mechanism. Such viewers are considered to be individual components, specified by a class name. This is a common approach in ClassMate: factory methods [6] will instantiate the specified class. This class must be a super class of the abstract viewer class contained in the framework. Although list and tree model viewers can be specified by a MIME type if all elements are of the same type, this is not chosen. These viewers also are specified by its class names.

3.2 Interactions

Interaction is an important aspect of the framework modeling, because it is an essential hallmark of education. LTSs are highly interactive systems and so, a design challenge lies into finding interactions which are unique and effective for the educational domain. However, to avoid the nightmare of an endless interaction collection, a suitable model should be able to catch new variants in a conceptually finite collection. Only in this way the essential interfaces, abstract and helper classes can be designed, necessarily for integration of new variants into the framework.

The modeling carried out represents such a finite collection. The modeling distinguishes data model, action type and view/control implementation. The latter refers to concrete implementations of abstract super types. Only two action types are defined: selecting and editing (something). An action is performed on a data model. I distinguish (similar to viewers) three different basic models: singleton, list and tree. Considering a particular data model and an action on it, several view/control-implementations are possible.

By modeling an action in the types 'edit' or 'select', one has to face some ambiguity. A select action is a single act. But an edit action is mostly a combination of different acts. For instance selecting a position, followed by changing something at that position (insert, delete, replace, reorder). Multiple ways to get the result are possible. Reordering can be done by dragging or by deleting and typing again. Deleting can be done by selecting the phrase and hitting a key, etc. If several differents acts may cause the same effect, then what exactly is 'edit'? First, almost any edit action implies an initial selection, but this is not the main act. Next, different acts follow and the effect of all these following acts is called 'edit': something will be changed. So, although 'edit' is not a precise definition in terms of the acts to be performed, the effect is clear. That is how it is used in the modeling, as 'change' opposite to 'select'. If needed, 'change' can be further specified as 'reorder', 'insert' and so on. In many cases such a need does not exist. This way the modeling also fits with the daily used terminology.

Similar to viewers, both the list and tree model contain singleton elements. This is a 'complication' because now acts may handle on the list or tree, as well as on the elements inside. In the modeling I keep these two apart. For example, an edit action on a list or tree is inserting, reordering or deleting elements. A select action on a list or tree will select an element. If however this element is *editable,* it may be followed by an edit action performed on the element 'inside' (see figure 5). This way more complex interactions are possible.

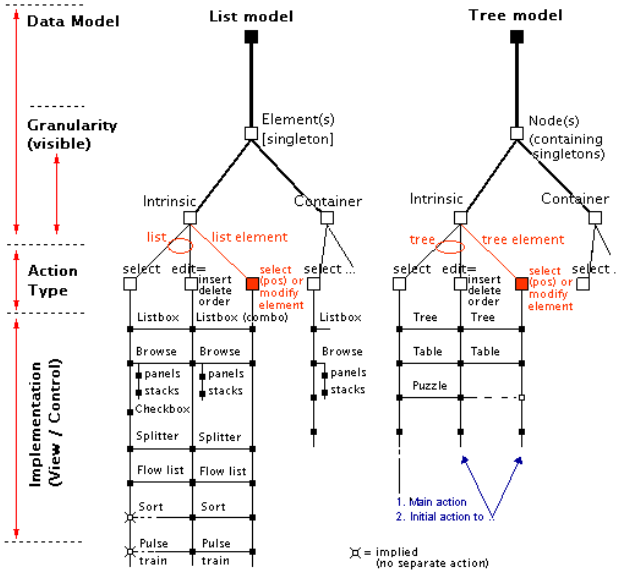


Fig.5. Interactions at list and tree data model

The framework contains a number of interaction components. This collection by can be easily extended by custom designed components, again using the base classes of the framework. An example of an interaction (screen fragment) is in the figure below.

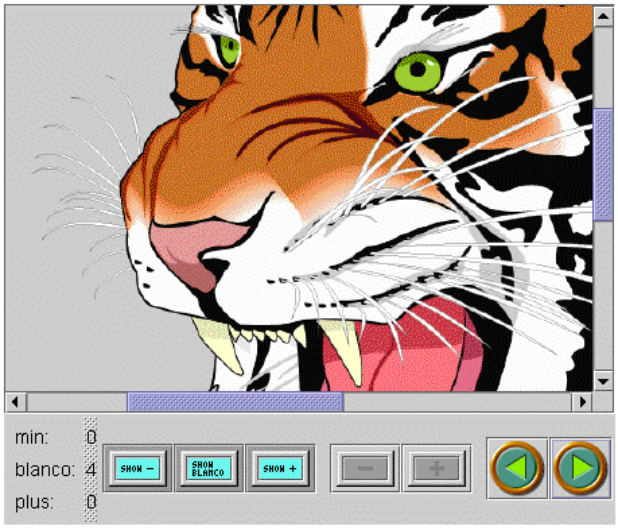


Fig.6. Interaction example: multiple-choice exercise as a browse variant. Options can be seen one by one and marked with 'minus' or 'plus'. Browse filters can be set.

4 Conclusion

ClassMate is not an integrated learning environment. It does not offer services for managing learner groups, storing individual profiles, retrieving learner progress and so on. Also the editing of item specifications is beyond the scope of the ClassMate framework. Nevertheless this is an important factor, because such editing is preferably within reach of the teacher using the program. Research [2,7] has shown that teacher involvement in development and maintenance is crucial for success: The teacher is the most prominent problem owner in the daily situation. In the pre-ClassMate period an effective working model for content creation was founded, suitable for teachers. It also turned out to be effective in the prototype phase. However, further exploration of this aspect is beyond the scope of this article.

The ClassMate design benefitted from the experiences, gathered at the development of dozens of computerbased lessons and testing programs in the past. One of the lessons learned is to make a distinction between educational designers and teachers. This still stands: there are no arguments to believe that non-technical teachers will be able to handle the ClassMate framework. So, educational designers are supposed to develop a session specification, because it presumes some technical knowledge and a sound knowledge of the ClassMate framework modeling.

The main achievements of the ClassMate application framework are rapid and code-safe application development by educational designers. On the basis of a specification, the framework is immediately functional. The framework is capable of delivering a broad variety of instruction types. For technicians, adding component extensions to the framework is straightforward.

References:

[1] IEEE Learning Technology Standards Committee, *Draft Standard for Learning Technology – Learning Technology Systems Architecture*, IEEE (http://grouper.ieee.org/LTSC/wg1), 2000.

[2] Verpoorten J.H., *A Model Based Approach to Courseware Development*, PhD Thesis, ECCO, Utrecht, 1995.

[3] Graaff R. de, *Differential Effects of Explicit Instruction on Second Language Acquisition*, PhD Thesis, HIL Dissertations, Leiden, 1997.

[4] Slagter P.J., *Learning by Instruction*, PhD Thesis, Rodopi, Amsterdam, 2000.

[5] Verpoorten J.H., Informatietechnologie voor studenten letteren. In Mirande M.J.A. (ed.) *De Kwaliteiten van Computerondersteund Onderwijs*, Coutinho, Bussum, 1994, pp. 318-331.

[6] Gamma E., Helm R., Johnson R. and Vlissides J., *Design Patterns: Elements of Object-Oriented Software,* ACM Press, New York, 1995.

[7] Blom J.J.C., *Use-oriented Courseware Development for Agricultural Education : An Ecological Approach*, PhD Thesis, Landbouwuniversiteit, Wageningen, 1997.