In Erik Duval, Ralf Klamma, Martin Wolpers (Eds.):
Lecture Notes in Computer Science 4753, pp. 86-100. Berlin, Heidelberg: Springer.

Making Sense of IMS Learning Design Level B: from specification to intuitive modeling software

Susanne Heyer[1], Petra Oberhuemer[1], Stefan Zander[2], Philipp Prenner[2]

University of Vienna
[1] Center for Teaching and Learning, Porzellangasse 33a, 1090 Wien, Austria
[2] Department of Distributed and Multimedia Systems, Liebiggasse 4/3-4, 1010 Wien, Austria
{susanne.heyer, petra.oberhuemer, stefan.zander, philipp.prenner}@univie.ac.at

Abstract. The IMS Learning Design (IMS LD) specification offers a language for modeling teaching and learning situations and flows. The specification contains great complexity, which represents a high entrance barrier to its use. To lower this entrance barrier to Learning Design, easy-to-handle software is needed that translates from the language used by instructional practitioners to IMS LD. This paper describes an approach for performing this translation. First, an analysis is described that was used for deriving typical uses of IMS LD Level B properties and conditions. Second, the resulting cases and translation transactions are presented. It is hypothesized that a wizard allows practitioners access to Level B functionalities even though the wizard reduces the complexity of the specification.

Keywords: IMS Learning Design, Level B, analysis, complexity reduction, Graphical Learning Modeler.

1 The specification IMS Learning Design

The IMS LD specification [1] prescribes a standardized modeling language for representing learning designs as a description of teaching and learning processes able to be executed by a software system that coordinates all involved people, resources and services. The specification hence supports the interoperability of learning designs aiming at enhancing sharing and re-usability of didactic settings.

The concept of the *learning activity* is central to the modeling language; IMS LD can thus be seen as an answer to the shortcomings of existing learning technology specifications focusing mainly on the sequencing of learning objects. Each learning activity is associated with learning objectives, prerequisites, a description and an environment. *Activities* and *environments* (consisting of resources and services) together with *roles* (e.g. learner) constitute the core *components* of IMS LD being managed by a *method*. The method uses the concepts of a theatrical play to orchestrate the activities. Within the method, the *role-part* connects roles to activities. Sequential *acts* containing the role-parts describe the teaching and learning flow,

whereas the transition from one act to the next serves as a point of synchronization. Acts finally constitute the *play*, which ends on completion of the last act. Refer to Fig. 1 for a visualization of components and method.

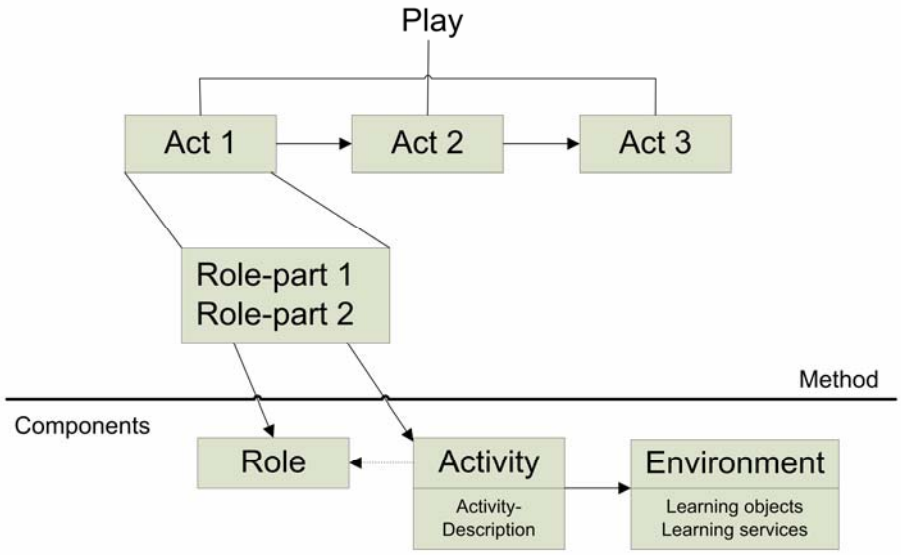

Fig. 1. IMS Learning Design model. [8]

In this paper, we are using the differentiation introduced by Britain [2] between the general concept of learning design, which refers to the design of learning activities and learning environments, using small 'l', small 'd', and the concepts in the IMS specification instantiated as units of learning[1] using capital 'L' and 'D' (Learning Design).

The specification has put forth three levels in order to enable phasing the software implementation efforts: Levels A, B, and C. Each of these levels offers an increasingly higher amount of detail and complexity for designing teaching and learning scenarios. Level A provides all basic elements for linearly sequencing activities and linking learning objects as well as services to activities. Although Level A leaves grey areas of interpretation [3], its principles can be well conceived. The more advanced mechanisms of IMS LD lie at Level B. While Level A mainly looks at the sequencing of activities, Level B allows for the structuring of individualized learning paths using *properties* to store data and *conditions* to act upon them. This means that learning objects and activities in a unit of learning can be adapted during runtime based on personal preferences or situational circumstances

[1] A unit of learning refers to a complete, self-contained unit of education or training, such as a course, a module, a lesson etc. [8]. From a technical point of view, a unit of learning is defined as an IMS content package that includes an IMS Learning Design [1].

like assessment scores of learners. Last but not least, Level C adds the possibility to automatically send notifications, i.e. messages, upon events that take place within the Learning Design. These notifications can be sent within the Learning Design or to external receivers.

Many researchers have proven that the IMS LD specification is indeed able to express a wide range of pedagogical approaches (e.g. [4]) showing that the specification is coherent and workable. The downside of the specification is, however, that only specialists, who have spent tremendous time working with the specification, are able to use it properly and to its full extent. For example, more than four years after the introduction of IMS LD only about 40 examples (as of June 13, 2007) of runnable units of learning can be found on the open access database DSpace [W001]. Most of these units of learning were added in 2005; only two units of learning were added in all of 2006[2].

While LD specialists feed the DSpace database with units of learning, the application of IMS LD by practitioners remains impeded by the complexity and technical nature of the specification [5]. The lack of authoring tools and runtime environments supporting the creation and delivery of Learning Designs represents yet another hindrance to a broad adoption of LD. As it cannot be expected from practitioners to familiarize themselves with the details of the specification, it is necessary to develop easy-to-handle tools that allow for an intuitive modeling process if we want them to apply the LD specification. These tools will have to support practitioners' approaches by hiding the IMS LD terminology and by translating the varied practical concepts into the rather rigid technical language of IMS LD.

In this article, we focus on describing the translation work to build a bridge from the IMS LD specification to intuitive modeling software in relation to Level B.

2 Tool perspectives and preparation of analysis

For capturing and sharing learning designs electronically, a functional and technical architecture is required allowing for the visual representation and interpretation of teaching and learning activities. Tools like the Reload Editor [W002] are not suitable for practitioners, since they require that learning designers know the specification and its functional concepts in detail. However, as the Reload software allows the use of *all* elements that the IMS LD specification defines, it provides a suitable basis for building a graphical modeling tool.

We thus developed the Graphical Learning Modeler (GLM) based on the Reload Editor, where the GLM provides an intuitive graphical user interface (GUI). The GLM encapsulates part of the complexity of IMS LD, its main functions being the interpretation of graphical Learning Designs and the translation to IMS LD code at Level A and partially at Level B including the identification of role-parts, concurrent activities, acts, activity structures, and dependencies among activities that require conditions. Learning designers are thus enabled to build Learning-Design-compliant units of learning without pre-knowledge of the specification.

[2] There was a third addition to the DSpace database in 2006; however, it contained presentation slides by David Griffiths and not an actual unit of learning.

The problem we faced was how to best present the more complex options at Level B of the IMS LD specification in a graphical modeling tool. Just looking at the IMS LD Information Model [1], it is next to impossible to immediately perceive the potential applications at Level B. Koper & Burgos [6] recognized this lack of perception and wrote about sample use instances of Level B functionalities. Practitioners are better referred to these concrete descriptions of uses, since a more common language and less IMS LD specific terminology is employed.

Increasing the perception by describing concrete applications of IMS LD at Level B as Koper & Burgos [6] have done serves two purposes: first, we are thus able to create a bridge between the language most practitioners use and the IMS LD specification, and second, we are thus able to derive rules for translating from typical applications to IMS LD code. In this regard, we answer the question of best representation by constructing software in wizard-structure, which will be integrated in the GLM and which guides the learning designers in creating interactive and individualized learning experiences. The wizard then represents the connection between practitioner-oriented language and the formal vocabulary offered by IMS LD.

As a first step of defining the wizard set up, we analyze Level B Learning Designs in order to describe typical uses of properties and conditions. As a second step, we make the correlations between typical uses and the setup of Level B's core concepts, namely properties, explicit. Based on these data, we are looking for translation mechanisms for the Graphical Learning Modeler wizard.

3 Analysis

In this section, we describe the analysis of units of learning to derive typical uses for properties. We see the property as the main component of Level B that is acted upon by conditions and monitor services. Within the analysis, we are thus focusing on properties in the first place, and put secondary foci on conditions and monitoring.

3.1 Set up

First, we took all publicly available units of learning – eighteen in number – that were conform to at least Level B of the IMS LD specification. These included units of learning from the DSpace database [W001], examples from the Best Practice and Implementation Guide [7], and the Learning Design Book [8]. We looked at the imsmanifest.xml[3] of these units of learning and, if available, at the source code of the accompanied resources of type imsld content[4]. From these files, we collected data

[3] The imsmanifest.xml is the part of the unit of learning content package containing the Learning Design.

[4] The IMS LD Information Model [1] states that objectives, prerequisites, learning objects, and activities are bound to be of type imsld content (XML files). Just like other resources (e.g. web content) imsld content can be referenced from a Learning Design but is not explicitly part of the IMS LD Information Model.

regarding the contained properties including their property-type, datatype, place of use, and function. The data was recorded in a spreadsheet application and then analyzed. The following questions guided our data analysis:

- What uses exist for properties?
- How does the type of property and its datatype relate to its use?

The total number of data sets collected approximated 800. This number does not equal the number of properties in the units of learning but the number of use instances of the properties, since one property defined in the components section (refer to Fig. 1) of the imsmanifest.xml could be used more than once within the Learning Design itself or in resources of type imsld content. For instance, the same property could be used in an eXtensible Markup Language (XML) file, where the user of the unit of learning will select a value for the property, and then be used in a comparison inside a condition to detect whether the choice the user made was correct. Each usage of a property, when possible to determine, was documented.

From the nearly 800 use instances for properties, we looked at a sample of 331 use instances more closely to draw correlations and to obtain a distribution for property functions. This sample was made up of property uses we could clearly describe and which could be included in the detailed analysis in a timely manner.

Problematic regarding the data recording were properties that were used within resources of type imsld content, which were not accessible to us. For instance, in the example units of learning of the Best Practice and Implementation Guide, resources of type imsld content were sometimes referenced in the imsmanifest.xml but not listed with their code. Accordingly, we were not able to determine the exact usages of the respective properties. Furthermore, some of the properties were essentially listed twice in our data set, since some units of learning received their structure from another, almost identical, unit of learning. For instance, the units of learning Quo Builder[5] and Quo Builder 2[6] are almost identical in terms of their property structures.

3.2 Results

The results of the analysis showed that almost all property-types defined in the IMS LD specification could be found in the sample units of learning. In part, strong correlations between certain property-types and datatypes could be identified. Yet, not all datatypes specified in the Learning Design specification [1] were represented in the sample units of learning. Also, not all functions provided by conditions had been applied in the sample units of learning.

3.2.1 Categorization of property functions

We found that from the property data sets we could derive six main categories of use instances:

[5] http://dspace.ou.nl/handle/1820/427
[6] http://dspace.ou.nl/handle/1820/428

- calculation
- change the value of a property
- manually determine the end of activities
- notifications
- show or hide components of the Learning Design
- view the value of a property.

Refer to Table 1 for a list of these main property functions including their subcategories, and descriptions.

Table 1. Categories including subcategories of property functions.

Main Category	Subcategory	Description
Calculation	none	Properties containing numerical values are used as operands in summation, subtraction, multiplication or division, or preparations thereof, for instance, counting points.
Change Value	in a resource of type imsld content	A property's value is changed through an according setup in a resource of type imsld content, inside a condition, or along with the end of an activity.
	on completion of activities	
	via condition	
Determine End	learning activity	A certain value of a property determines when a learning or support activity is over.
	support activity	
Notification	none	Used to send notifications to users taking part in the unit of learning to inform them of certain happenings in the unit of learning such as the ending of an activity.
Show or Hide	activity structure	Used in a condition, this function checks if properties have reached a certain value. Upon that value, components of the Learning Design such as activities or environments can be shown or hidden.
	class[7]	
	environment	
	learning activity	
View Value	none	The value of a property is made visible by placing it in a resource of type imsld content that is used in the unit of learning.

The categories introduced in Table 1 are interrelated. For instance, for a property to be viewed, its value was usually changed in advance. The change, however, could take place through a calculation inside a condition or through an input that a user of the unit of learning provides during runtime. Also, the Show function and the Hide function often appear together in the imsmanifest.xml, meaning that an activity will be hidden until a certain property value has been reached and only then is the activity being shown to the learner.

Fig. 2 shows the distribution of the 331 sample property use instances according to the main functions described in Table 1. For the identified functions Change Value,

[7] The attribute class is a global attribute, which was defined by the World Wide Web Consortium initially for XHTML use in cascading style sheets [W005].

Show/Hide, and Determine End, their corresponding subcategories have been listed separately as a portion of the total count.

Looking at Fig. 2, Change Value is the most frequently used function with properties among the sample we took. This is not surprising since it can be regarded as the primary function of properties: the intention of properties is that values are input and stored. Change value thus represents the possibility to provide input and have this input stored. All other functions except for Notification are usually consequences of the Change Value function of a property.

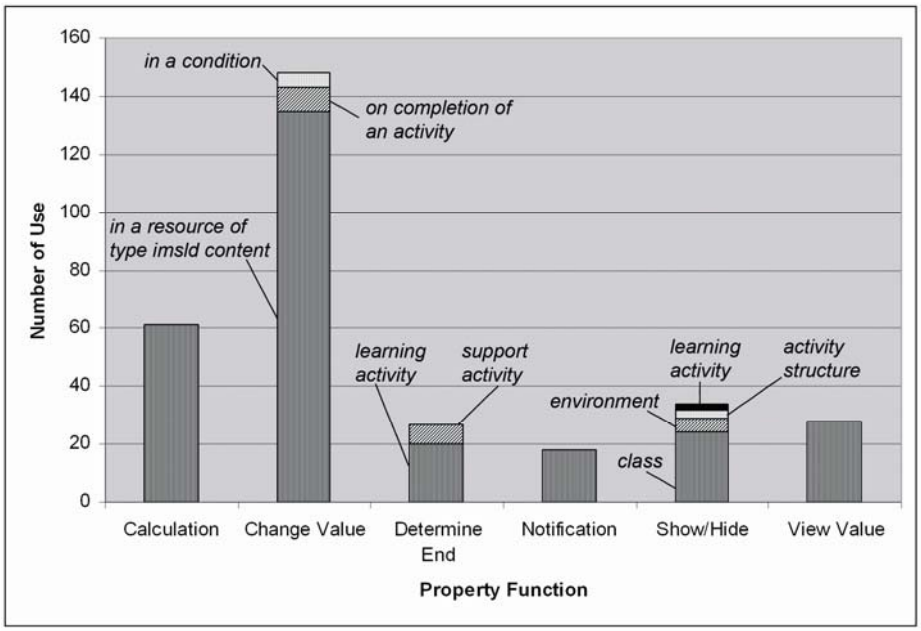

Fig. 2. Distribution of property functions including subcategories.

The values of properties are most often changed inside resources of type imsld content. This is apparent when looking at the subcategory "in a resource of type imsld content" of the Change Value function in Fig. 2. Properties set inside imsld content resources allow learners to provide own input and interact with the system. For instance, learners can only provide feedback or give answers through interaction with a property that is placed inside a resource of type imsld content. Only through properties is this type of interaction and personalization in Learning Designs possible.

For a change of a property value inside a resource, the global element set-property is used. Implementing the set-property element requires that the learning designer knows XML-coding, since the set-property element is included in the XML content schema using XML namespaces. This means that even if we use the Reload Editor, which takes over the part of coding the LD conforming XML, learning designers would still need XML-coding skills to write the accompanying imsld content.

The same is true for the Show or Hide function (refer to Fig. 2): most of the time, the class attribute was used to show or hide portions of text (e.g. differing feedback depending on a result or choice made by the learner). Thus, only if the learning designer knows XML, s/he is able to employ such personalized elements in the Learning Design. This is a major demand of the LD users from our point of view.

The View Value function is possibly underrepresented in the distribution of our sample in Fig. 2. We explain this misrepresentation since in some units of learning a property's value may only have been changed once but was made visible in different resources of type imsld content or in several classes. Furthermore, a *change* of a property's value inside a resource may be inferred and thus the data recorded if the property has no other defined purpose within the imsmanifest.xml, even if the resource's source code is not accessible. *Viewing* a property's value, on the other hand, may not as easily be concluded, since the property could just be carrying needed information in the background of the Learning Design. Determining if a property will in fact be *viewed*, i.e. shown to a role, can only be achieved by looking at the source code of imsld content. We thus attribute this potential misrepresentation in Fig. 2 to time constraints in checking all imsld content and the non-accessible codes of imsld content resources that were merely referenced in the imsmanifest.xml.

We are aware that the distribution shown in Fig. 2 is not representative of all the properties in the Learning Design example units of learning. The units of learning differ highly in the number and type of properties they employ. For instance, the unit of learning "Learning to Listen to Jazz" [9] makes use of 108 properties, 23 property groups and one property-type. On the other hand, the unit of learning "Programmed Instruction" from the Best Practice and Implementation Guide [7] only uses two properties, no property groups and one property-type. Thus, including one unit of learning and excluding another may dramatically change the distribution.

3.2.2 Correlations between property-type and datatype

Of the five types of properties that IMS LD allows (local, local-personal, local-role, global-personal, global), the examples we analyzed used all of these but the global property-type. The most commonly used datatypes were boolean[8], integer[9], string[10] and uri[11]. The datatypes text and file were also used, but considerably less often than the formerly named datatypes. We couldn't find usages of the datatypes datetime[12], duration[13], other[14], and real[15] within our sample of the example units of learning.

The analysis of the properties showed that there are typical correlations between the type of property being used, its datatype and the concrete application. An example for this is that all properties used for the category Calculation (cp. Table 1) were of property-type local-personal, and had the datatype integer. Their concrete use was, for

[8] Represents binary logic, e.g. on/off, true/false.
[9] Represents whole positive and negative numbers.
[10] Represents legal character strings, no longer than 2000 characters.
[11] uri is short for Uniform Resource Identifier.
[12] Specifies a date and time in a specific format.
[13] Specifies an amount of time in a specific format.
[14] Random datatype without specification.
[15] Represents arbitrary precision decimal numbers.

example, to calculate points of achievement. Another example is that the end of an activity is often determined by a property of type local-personal with datatype boolean. Local-personal property is here used so that each learner can individually determine the end of the activity, while the boolean datatype is used because the activity can only have two status: finished or not finished.

Notifications, which are only of concern at Level C, are not in our immediate focus since we first aim at implementing Level B in the Graphical Learning Modeler. Nevertheless, properties that are used with notifications are always of type global-personal and use the datatype uri (in our analysis usually the email address of the tutor). Among all units of learning, global-personal properties were always 'existing href' (due to their conceptual nature of working across units of learning).

Information collected from such correlations between uses of properties, property-type and datatype, builds the basis for our wizard design to be integrated in the Graphical Learning Modeler. The wizard design will be further described in section 4.

3.3 Limitations of property use

Within the results of the analysis, we pointed out a few limitations of both the example units of learning and technical setup of the IMS LD specification. Regarding the latter, the fact that the value of properties is most often changed inside resources of type imsld content (XML-files) and that the attribute class is mainly used for showing and hiding text surprised us. Consequentially, the learning designer must possess the ability to construct valid XML-files in order to implement personalized or interactive elements in a Learning Design. To work around this, we have to develop new use concepts that are still able to provide these functionalities, yet, at the same time, hide the XML-specific terminology. These particular requirements have not yet been considered in the design of the wizard-structure for the Graphical Learning Modeler described herein but will be subject to further developments.

In addition to the limited use of datatypes for properties, only a portion of the possible condition operations was used in the sample units of learning. Examples for expressions that were not used in any of the units of learning are users-in-role, complete-an-act reference, time-unit-of-learning-started, and current-datetime. Therefore, we were not yet able to identify typical uses for these expressions.

When deriving the set of rules guiding the development of the Graphical Learning Modeler wizard, we focused primarily on those functionalities we were able to detect in the analyzed applications. More concrete descriptions of these translation efforts are described in the following section.

4 Projecting analysis results onto development of Level B wizard

In order to derive useful input for designing a wizard to be used in the GLM for Learning Design, we are correlating three types of data:

1. the concrete applications of properties described in the sample units of learning (like "students place a written comment on a controversial topic into a text box")

2. the categorization we established in the analysis (cp. Table 1, for instance, Change Value, Calculate etc.), and
3. the correlation between property-type and datatype.

For each application that we identified in a unit of learning (1.), we related the property-type and its datatype (3.) as well as the place of use of the property (2.) to the application in order to find typical uses and to derive rules where appropriate.

From the first data, we can derive what the learning designer might expect to see, since the specific examples comprise activities or procedures that teaching practitioners can relate to in their own terminology. The second data in relation to the first will inspire the user interface setup (for instance, showing and hiding are displayed together in the GUI since they are conceptually and technically related). Using the third data and relating it to the first two, we are able to determine what code must be written upon the choice that the user makes inside the wizard. These interrelations will be described in the following sections.

4.1 Combining property applications and property functions

Taking the example units of learning, we found different applications of properties. Examples of uses are “writing a comment regarding a controversial statement”, and “writing an answer to a short essay question”. Even though practitioners are best able to understand this type of concrete description, for the design of a wizard it would not be advantageous to create lengthy lists of descriptions with very specific use instances.

Instead of providing each concrete use its own place in the wizard, we are attempting to group the uses sensibly and create a name for that group on a first abstraction level. For instance, the two examples given above (writing a comment and writing an answer to a question) are highly related from a technical point of view within IMS LD. Conceptually on a teaching practice level, they are related as well. Although these uses represent two different cases, we are able to group them on a first abstraction level to “writing a short text”. Even one more abstraction level above, we would call this “giving textual input during runtime”. “Giving textual input during runtime” would then be a subgroup of “giving input during runtime”. Via inductions like this, we are able to construct a decision tree that starts at more abstract levels and as you go into the tree, become more specific.

In a next step, we have to combine the so developed decision tree with the technical functions that IMS LD allows. For this, we mesh the decision tree with the categorization of property functions (cp. Table 1) and again draw correlations. Using this technique, we were able to derive four main categories that represent the entrance level for our Graphical Learning Modeler wizard. In combining the conceptual uses with the functions of properties as listed in Table 1, we created the following functional entrances:

1. Show/Hide (subsumes View Value and Show/Hide functionalities)
2. Providing Input Possibilities (subsumes Change Value functionalities)
3. Points (subsumes Calculation functionalities)

4. Administration and Control (subsumes Determine End of Activities, and grouping of Input Possibilities)

Each of these four main choices contains several substeps that get more and more specific. For instance, the second menu option “Providing input possibilities” opens the options of choosing either a “Choice from predefined values” (opening a drop-down box at runtime to select, and therefore input, values) or “Free input” (a space is provided at runtime for the learner to enter any text or numbers). Upon choice of the former, there is an option of differentiating between “multiple choice” (which in the background of the GLM employs the datatype “string” as well as the restriction type “enumeration” for properties) or “vote” (which in the background uses the datatype “boolean”). Providing these choices at the time of design, the learning designer is thus able to include interactive elements and individualized learning paths in the unit of learning. Whenever there is additional input needed, for instance, if the learning designer specifies the values that are to be chosen among like answers to a multiple choice question, the GLM provides a dialog that will ask for this information in common language.

4.2 Wizard setup for determining property-type and datatype from use

As we have shown in the analysis, the type of property (local, local-personal etc.) often had strong correlations with the application and datatype being used. Because of this, we felt reassured that developing a wizard-like structure is a manageable approach, since we are able to either directly derive the required information from the correlations or to limit the questions we need to ask the learning designer. The wizard’s main function is to guide the learning designer through a set of decision steps, which produces IMS LD conformant code in the background. This way, the learning designer will not be faced with IMS LD specific terminology.

For example, if the learning designer picks from the wizard (via several steps) the option to “show this activity only when the learner has reached a certain number of points”, then we know that we have to construct (if it is a new property) or refer to (if the learning designer already established a property for counting points) a property of type local-personal with datatype integer. We then have to refer to this property within a condition, which will observe whether the property has reached the specified value (which we ask the learning designer to determine in a dialog; e.g. five points must be reached). The condition will then have the following setup (only relevant portions of the code are being shown):

```
<if>
    <is>
          <property-ref ref=”points”>
          <property-value>5
<then>
    <show>
          <learning-activity-ref ref=”activity-name”>
<else>
    <hide>
          <learning-activity-ref ref=”activity-name”>
```

As the learner, who will go through the described learning path, collects points, s/he will only see the specified activity once s/he has collected the necessary points. Since for every learner the system detects individually whether they have reached the five points, we are using the property type local-personal for this purpose. Since points are usually whole numbers that are to be added or calculated, we are using the datatype integer.

For every path that can be selected within the wizard, we have developed functionalities that follow the herein presented schema. Thus, we derived rules how to set up the imsmanifest.xml from the choices that the learning designer makes using the wizard.

At this point, we are at the conceptual stage of this development. However, we are implementing this wizard into our existing Graphical Learning Modeler.

4.3 Challenges for the Graphical Learning Modeler

We are aware that a wizard designed in the way we just described allows the construction of *typical* uses for properties and conditions. For application of a greater number of Level B functions beyond these typical use instances, the learning designer would have to be able to use the LD terminology. In further developments, we strive to reduce this gap between the available functions of the wizard and the potential functions offered in the LD specification.

Despite the restrictions that the wizard possesses on the one hand, on the other hand, it finally offers practitioners access to IMS LD and places them in a position to design Learning Designs including functions of properties without any knowledge of XML or IMS LD: Just as users of wysiwyg[16]-HTML-editors do not need to know coding procedures in HTML, so do users of a Graphical Learning Modeler not need to know Learning Design specific coding procedures. This is the advantage that we see in developing the GLM.

Suppose that the provision of a Graphical Learning Modeler leads in turn to a higher adoption rate of the IMS LD specification – only then would the learning design community be able to judge whether the specification truly fulfils its promise of supporting the exchange and re-use of Learning Designs [1].

5 Technical Setting

The GLM makes substantial use of functionalities implemented in the Reload Learning Design Editor and extends them with additional information and business logic necessary for visually representing IMS LD elements such as learning activities and their dependencies among each other.

Our decision to use the Graphical Editing Framework (GEF) as the graphical toolkit and framework for modeling teaching and learning workflows on top of the

[16] wysiwyg is short for What You See Is What You Get.

rich client platform Eclipse [W003] was driven by the results from a previously conducted evaluation of existing modeling frameworks. In this evaluation phase, considerations had also been given to the Graphical Modeling Framework (GMF), which provides suitable functions for building graphical modeling tools. The main difference between GEF and GMF lies in the conceptual orientation of GMF, which is more suitable for building Unified Modeling Language (UML) models. GMF also requires the underlying data model to be conform with the Eclipse Modeling Framework (EMF) data model specification. The Eclipse platform and GEF [W004] are both open source technologies and are ideally suited to provide the technical basis on which the Graphical Learning Modeler is built.

The Graphical Editing Framework provides capabilities to easily develop rich and adaptable visual representations of existing data models. These editing capabilities allow for the creation of graphical editors for almost any arbitrarily complex model as well as user interactions on this model. Modifications on the underlying data model based on user interactions, such as changing element properties or changing the model structure, are supported as well as performed by using common functions such as drag and drop, copy and paste, and actions invoked from menus and toolbars [10]. The underlying model will be updated accordingly.

To display graphical elements called widgets, GEF makes use of the Standard Widget Toolkit (SWT), which acts as a bridge for accessing and presenting GUI elements from the underlying operating system. The main advantage of SWT over existing and related technologies such as the Advanced Window Toolkit (AWT) or Swing is its seamless integration into the existing working environment since SWT – in contrast to other technologies – does not emulate the graphical interface. It rather acts as an adapter or bridge to the GUI-Elements and Services of the operating system.

The use of SWT as a mechanism for representing graphical elements also provides enhanced interaction with the prevailing working system components while making better use of existing system resources. The disadvantage resulting from the usage of SWT is the loss of platform neutrality and independency, since the application needs to be compiled for any target operation system separately.

Due to the fact that GEF is fully compliant with the Model-View-Controller principle[17], it provides a basis for extending and improving the existing functionality while leaving the domain models untouched. The separation of business logic, visual presentation, and the domain model allows a great form of flexibility in extending the functional range of the Graphical Learning Modeler where additional functions can be added without much effort.

Fig. 3 depicts the Graphical Learning Modeler: activity and environment lists as well as design functions are seen in the upper left, learner and staff roles in the lower left, the graphical workspace holding the activity sequence in the upper right, and a provisional “check” window in the lower right. The check window is used during

[17] Interactive applications, which follow the Model-View-Controller architectural approach, are divided into three layers (model, view, and controller) and decouple their respective responsibilities. Each layer handles specific tasks as well as has specific responsibilities and interdependences to other layers and their respective components. This form of separation among model, view, and controller objects reduces code duplication and eases the maintenance of applications.

development to quickly verify that the GLM interprets the graphical layout in terms of IMS LD correctly. The Level B wizard for the more complex functions as presented in the analysis will be integrated as part of the design functions in the upper left among operations like selection point or deliberate synchronization. Level B functionalities, which the GLM is able to interpret directly from the graphical layout, will be automatically translated without input from the learning designer.

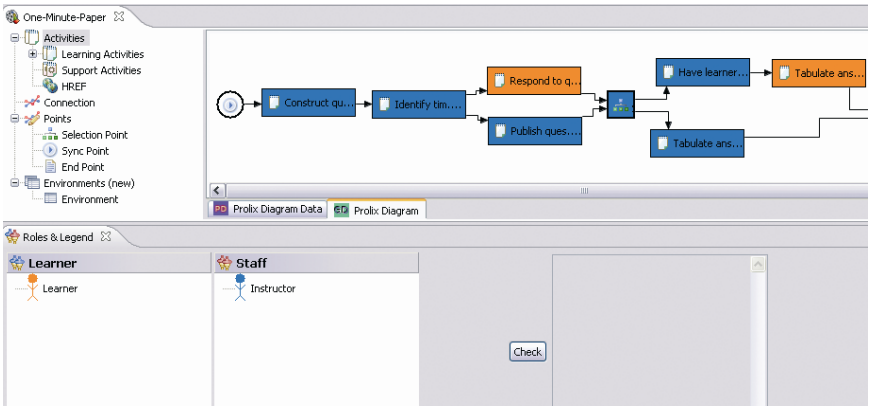

Fig. 3. Screenshot of the Graphical Learning Modeler.

The GLM encapsulates the actual data components implemented in the Reload Editor and complements them with additional information to build its own model on which the visual representations are generated. Reload data components and business logic is extended with GLM-specific data and logic to provide the desired functionality. The Graphical Learning Modeler therefore acts as an additional functional layer on top of Reload incorporating its own business logic and data model for generating visual representations of IMS LD elements. The use of the graphical framework GEF allows for the uncomplicated extension and customization of this functional layer with additional functionalities.

6 Conclusion

In this article, we introduced a system for providing Level B properties in a translated form to users of a Graphical Learning Modeler. We used publicly available samples of units of learning to derive typical uses of properties. The analysis showed that only part of the potential that Level B holds has been expressed in publicly available units of learning. This fraction, however, can be translated into typical uses and placed into a structure, which gives practitioners of IMS LD access to the functions of Level B properties and conditions.

Acknowledgments. This article was written in the context of the research and development integrated project PROLIX, which is co-funded by the European Commission under the Sixth Framework Programme "Information Society Technologies". The Graphical Learning Modeler that is being discussed in this paper is a development of the University of Vienna with main contributions from Maia Zaharieva at the Multimedia Information Systems Group headed by Wolfgang Klas.

References

1. Koper, R., Olivier, B., Anderson, T. (eds.): IMS Learning Design Information Model. IMS Global Learning Consortium (2003)
2. Britain, S.: A Review of Learning Design: Concept, Specifications and Tools. A report for the JISC E-learning Pedagogy Programme (2004)
3. Oberhuemer, P., Heyer, S.: Probleme bei der Umsetzung didaktischer Modelle in IMS Learning Design: eine Anwenderperspektive. Zeitschrift für e-Learning, Lernkultur und Bildungstechnologie. (2007) in press
4. van Es, R., Koper, R.: Testing the pedagogical expressiveness of IMS LD. Journal of Educational Technology & Society, (2006) 9 (1) 229-249
5. Beetham, H.: Review: developing e-Learning Models for the JISC Practitioner Communities. Version 2.1. Joint Information Systems Committee e-learning and Pedagogy Programme (2004)
6. Koper, R., Burgos, D.: Developing Advanced Units of Learning Using IMS Learning Design Level B. International Journal on Advanced Technology for Learning (2005) 2(4) 252-259
7. Koper, R., Olivier, B., Anderson, T. (eds.): IMS Learning Design Best Practice and Implementation Guide. IMS Global Learning Consortium (2003)
8. Koper, R., Tattersall, C. (eds.): Learning Design: A Handbook on Modelling and Delivering Networked Education and Training. Springer, Berlin, Heidelberg (2005)
9. Tattersall, C., Burgos, D.: Learning to Listen to Jazz. (2005) Retrieved March 26, 2007, from http://dspace.ou.nl/handle/1820/371
10. Moore, B., Dean, D., Gerber, A., Wagenknecht, G., Vanderheyden, P.: Eclipse Development Using the Graphical Editing Framework and the Eclipse Modelling Framework. IBM Red Book: ibm.com/redbooks (2004)

[W001] http://dspace.learningnetworks.org/handle/1820/16/browse-title
[W002] http://www.reload.ac.uk/ldeditor.html
[W003] http://www.eclipse.org/
[W004] http://www.eclipse.org/gef/
[W005] http://www.w3.org/TR/REC-html40/struct/global.html#h-7.5.2