INTERACTIVE LEARNING OBJECTS: A FRAMEWORK BASED APPROACH

Friedbert Kaspar
University of Applied Science
Faculty of Computer Science
D-78120 Furtwangen

ABSTRACT

A rapid content development approach to develop interactive Learning Objects for the support of programming education is presented. The identification and exploit of typical interaction pattern in the learning scenario as concept behind the approach is explained. The architecture, design and implementation of the developed Software system are illustrated. The status, experiences made and future plans are reported.

TARGET/PHILOSOPHY

With the Codewitz project the EU supports the development of visualization drills to improve the programming education. We joined this project to enhance the self directed learning part of our students by producing interactive learning objects (LOs). From past experiences we formulated the requirements for our approach of material production. The production of material has to be efficient. The teacher must have full control of the used material. That means, if she is not satisfied with some formulation she has to be able to change this easily and fast. Navigation through the material has to be supported appropriately. The material must be accessible and usable by a Web browser. Our approach is guided by the idea to use concepts of component based software development in the context of the development of LOs. Our approach is to develop a framework for the specialized product line oriented rapid content development [BIK]. We are convinced that this is more efficient than using a general purpose content development tool because special characteristics of the domain, i.e. programming education are exploited.

CONCEPT - IDENTIFY AND IMPLEMENT TYPICAL INTERACTION PATTERN

The idea we started with was to identify typical interaction pattern in the interaction of the instructor with the student when teaching a programming language. With table 1 we tried to develop a systematic description of types of interaction. The table shown is not intended to be complete, but should motivate to add on.

To explain some syntax element typically starts with a programming example. This example is then stepped through (more or less) line by line. This we mean by the type StepByStep. Clearly a type may have subtypes. For example another StepByStep type may be added with several steps of explanation per line of code, for example to illustrate the evaluation of expressions. There may be specialized StepByStep types with audio tracks, i.e. oral explanations connected to a step. The StepByStep approach may be distributed in several windows, for example to explain functions.

Type	Description
StepByStep	*Step by Step explanation of Code, works like a debugger.*
Type 11	Input fix, Output fix, max. one step per line
FindError	*locate error in source code*
Type 21	1. Step: find erroneous line, 2. Step: Dialog with different possible explanations for the error, max. 2 Hints, Solution button
ProgramOutput	*Deals with program output*
Type 31	For a given output the user has to find the adequate initialization of a variable.
Type 32	User initializes Variable, finds the correct output from a collection of (five) answers
Type 33	A variable is initialized randomly and the user has to find the output from a List of answers
Type 34	A variable is initialized randomly and the user has to type in the output of the program
Type 35	Program is fixed, user gives output of the program in text input dialog, hints available, solution available
Type 36	Program is fixed, user gives output of the program in check box dialog, hints, solution
CompleteProgram	*A missing line of code has to be added.*
Type 41	A line of Code is missing. The user has to point to where to insert the line. A list of 5 possibilities is given to chose from.
Type 42	A line of Code is missing. The user has to point to where to insert the line. The user has to type in the missing line

Table 1: Types of interaction for Learning Objects

Besides giving step by step explanations to a student we challenge the student by questions to check the level of knowledge and to focus on especially important facts. In programming education this is typically done by the questions: Where is the error? What is the output of the program? As a step towards writing programs we ask the student to complete a program, by adding a missing line. These interactions we tried to describe in table 1. This can be considered as the use cases to start with for the development of our application.

As stated above the list of types in the table 1 is not complete. Therefore when developing the architecture of our application we have to take care that new types may be added easily, without affecting existing code. We want to reach this goal by the appropriate usage of design pattern.

APPROACH

Our goal was to develop an expandable framework with a reference implementation. We preferred open standards and public domain software libraries and tools. For the software architecture we let us guide from the Model-View-Controller (MVC) paradigm [MVC]. The requirement that the LO has to be displayed in a Web-browser, resulted in the decision to use Java and the applet technology. The user interface is realized using Swing. The data handling is done by means of XML [XML].

IMPLEMENTATION

Class design

o make the system flexible, i.e. to make it easily expandable by new types of LOs an suitable design has been chosen. Appropriate design pattern [GOF],[GRAND] have to be used, which allow to extending the existing code without changing code of already implemented types. Generally speaking, design pattern are used to support code reusability. The central pattern suitable in this case is the factory pattern. This means that a factory is instantiated when starting the program. This factory produces instances of a concrete type. The associations between the lesson controller, the lesson model and the lesson view (LessonDialog) as well as the interfaces of the classes are defined on the abstract level. Accordingly the only place where code has to be added to existing code when adding a new type is in the LessonFactory. See Fig. 1 for an excerpt of the class diagram. We aim at reaching for our software the quality of a framework with a reference implementation.

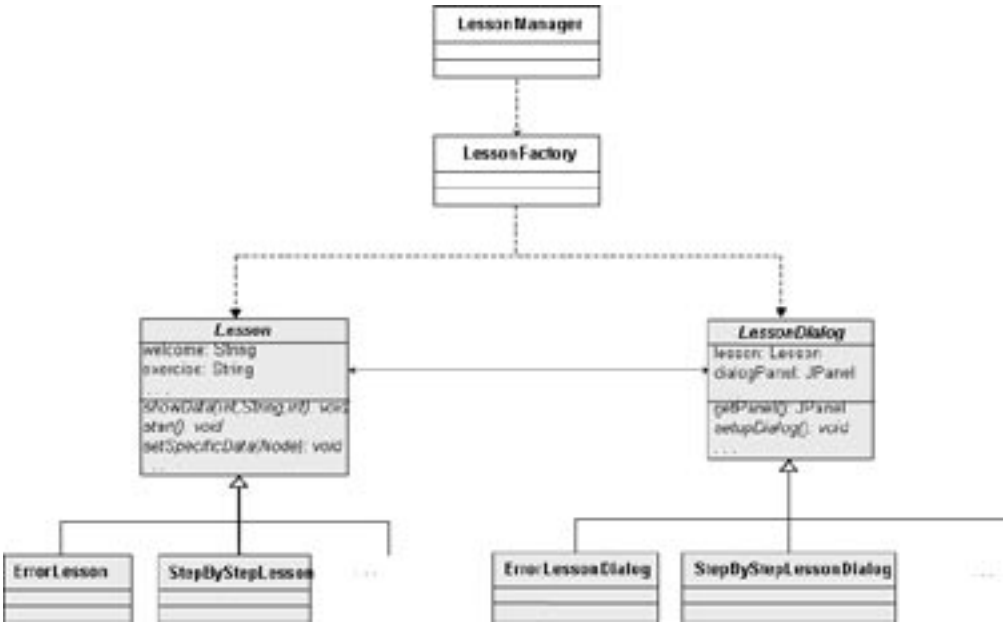

Figure 1: A section of the UML diagram, to describe the class design of the CLearning framework

Data Model

The requirement to produce LOs with different types efficiently, leads to the separation of the LO description and the implementation of a type. A type is represented by the definition of a Schema and an LO is an instance of the Schema. The method chosen is XML, i.e. the Schema is described by XML Schema and an LO is an instance of the related Schema [XML]. To illustrate the structure, in Figure 2 the XML Schema of type 21 is shown graphically. By the tree structure of the Schema it is easy to add more specific features to an LO type, that means to describe subtypes of an existing type. Having a new type normally means to also develop an appropriate controller and viewer for the type. On the other hand several views with eventually modified controllers may exist for the same Schema.

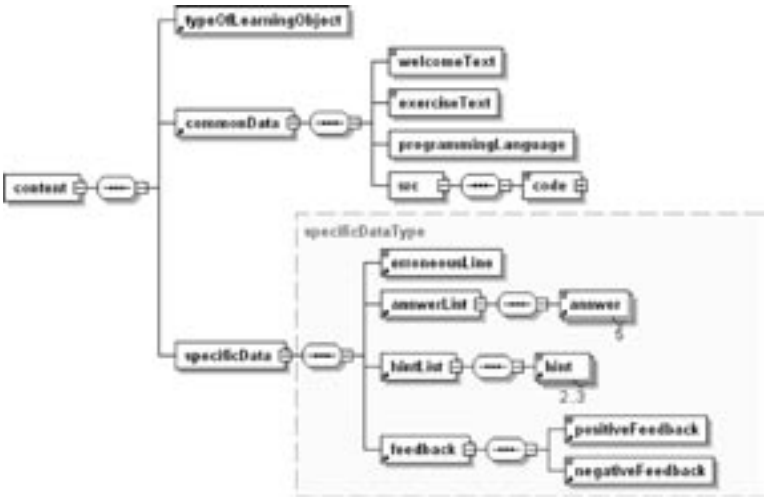

Figure 2: XML Schema for the type 21.

By the text file nature of the XML Schema and XML descriptions it is easy to understand, change and to produce new LOs by simply using a text editor. It is however for convenience or to increase productivity also possible to use a specialized XML editor like XMLspy [ALTOVA] or the XML plug-in for eclipse [XMLPLG]. Performance is not an issue, so the advantage to have a text representation outbalances in any case the advantage of faster data access of a binary data representation.

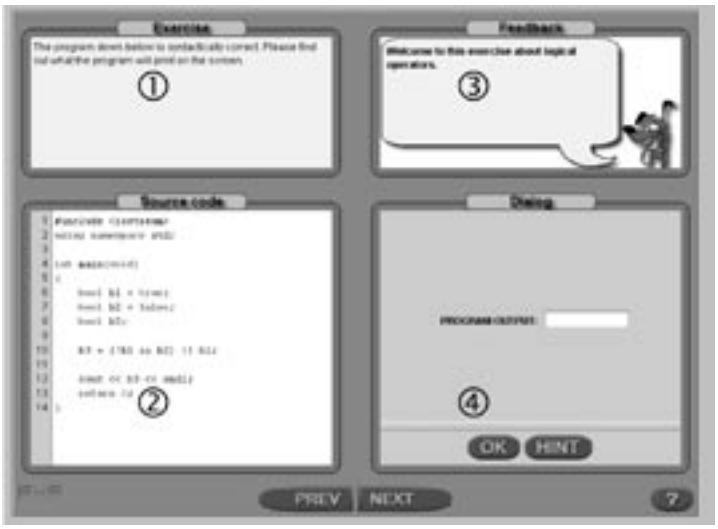

Figure 3: Layout of user interface

User interface

The current user interface consists of four dialog windows, (1) the explanation of the exercise, (2) a view on the source code of the example, (3) the feedback window, where the LO may display the reaction of the LO to an input of the user and (4) the dialog window, where the user of the LO may input a reaction to a challenge of the LO or controls the LO. The structure is currently the same for all types of LOs. However these windows behave differently for different types of LOs. A future plan is to get more flexibility in arranging and configuring the windows. For example it is planned to have a choice between different skins.

Status, Experience

For all major types at least one subtype has been implemented [SILARD]. A stable version of the software has been reached. A comprehensive collection of LOs has been produced to gain experience with all implemented types [PANZER]. Together with the LOs from the Codewitz Material Bank [CDW] a basic course in C++ [KAS] is essentially supported by LOs. Evaluation of the usage and acceptance of the LOs by the students is in progress. The effort to produce LOs of an existing type is dominated by the effort to develop the idea and to describe the design. The implementation time, i.e. the time to describe an LO with XML is small compared to this effort and insofar sufficiently small. Implementing new types challenges the reusability of the existing code. With student programmers it is important to defend the code. We observed that it is very difficult to keep the code clean over a longer history. The structure begins to decay after two or three generations of students. There is some need for action. To speak with Andrew Hunt the challenge is not to allow broken windows [HUNT]. Therefore we started recently with a test driven development approach [BECK]. It is our hope, that this is the appropriate way to improve the software quality under the existing conditions.

Future plans

Clearly we plan to produce more LOs for existing types and naturally we want to identify, describe and implement new learning scenarios. A near goal is to fully cover a basic course in C++ by LOs. Beyond that with little work all existing types can be made ready to produce LOs for a Java course. Support for other programming languages can be added. To implement support for internationalization is interesting for a broader distribution. An important issue is Metadata, to support the management of a growing number of LOs, to keep track of versions, to improve the usability for the teacher and the student, etc. Our XML Schema description is prepared for the usage of the IEEE LOM standard [LOM]. Integration of the LOs into Learning Management Systems (LMS) by implementing an appropriate interface could help the teacher to keep track of the student activities. An idea to increase the flexibility of the framework is to implement a service plug-in structure. A service plug-in is a parameterized Java class with a standardized interface which is referred in the XML description for the LO and which is loaded at runtime when needed. This supports the implementation of new, more flexible types.

REFERENCES

[ALTOVA]	XMLSpy: http://www.altova.com/
[BECK]	Kent Beck; Test-Driven Development; Addison Wesley; 2005
[BIK]	http://www.aifb.uni-karlsruhe.de/BIK/vpa/Pankratius,%20Oberweis,%20Stucky, %20E-Learning%20Lernobjekte.pdf; 2005; see also References included
[CDW]	http://www.codewitz.net
[ECLIPSE]	http://www.eclipse.org
[GOF]	Erich Gamma, Richard Helm, Ralph E. Johnson; Entwurfsmuster Elemente wieder verwendbarer objektorientierter Software; Addison Wesley; 2004
[GRAND]	Mark Grand; Pattern in Java, Volume 1; Wiley; 1998
[HUNT]	Andrew Hunt, David Thomas; Der Pragmatische Programmierer; Hanser; 2003
(KAS]	http://www.hs-furtwangen.de/~kaspar/
[LOM]	IEEE Computer Society, IEEE Standard for Learning Object Metadata, IEEE Std P1484.12.1TM-2002, September 6, 2002
[MVC]	Frank Buschmann et al; Pattern-Oriented Software Architecture, Volume 1; Wiley; 1996
[PANZER]	Florian Panzer; Learning Objects für C++; Diploma Thesis; Furtwangen University; 2005
[SILARD]	Julius Christian Silard; Entwicklung eines Frameworks zur Erstellung von Learning Objects. Diplomarbeit; Furtwangen University; 2005
[XML]	http://www.xml.org
[XMLPLG]	http://download.eclipse.org/webtools/updates/