ADRIAN
A PLATFORM FOR E-LEARNING CONTENT PRODUCTION

JOSÉ CARLOS RAMALHO AND GIOVANI LIBRELOTTO AND PEDRO HENRIQUES

Department of Informatics, University of Minho, Campus de Gualtar, 4710-057 Braga, Portugal.
E-mail: jcr@di.uminho.pt, grl@di.uminho.pt, prh@di.uminho.pt

Universities and other institutions related to education are investing time and resources in E-learning initiatives. This leads to an increasing number of course offers in E-learning format. There are environments, called *Learning Management Systems* (LMS), designed to help teachers in the management of their courses. These systems support the management of administrative information, student evaluation and all the interactivity between teacher and students and among students. However they do not provide tools to help teachers to prepare and to produce content: lessons, tests, guided lab sessions, ... Here is where ADRIAN comes into the scene providing support for content production.

ADRIAN is composed by several components: one component to help producing lessons and lab guided sessions; one component for the production of tests and exams; one component to support the production of multimedia presentations; and one component to generate interfaces that integrate all the material produced (content parts) by the other components or developed elsewhere by the teacher.

The whole system is being developed with XML (eXtended Markup Language) using descriptive markup for content, and related technologies like XSL (eXtended Stylesheet Language) for content transformations. This way we ensure the portability and platform independence of the system.

The last mentioned component, the integration component, is based on ontologies; the user is asked to define an ontology for his course. After that the system generates automatically the web interface that integrates all the courseware components.

In this paper we describe the ADRIAN architecture and the components developed so far. This description is illustrated with a real case study.

1 Introduction

Today the web provides an excellent channel to distribute information and to access it. Its application in learning environments was a question of time. Today is the reality.

The first experiences we have made of using the web for teaching purposes date back to 1994. Back then we were using mailing-lists to interact with students and web pages to make available all kinds of information.

Nowadays the demand for E-learning courses and materials is growing everyday. The possibility of being able to follow a course from the comfort of one's home is attractive to many people that can not be present at the time the lectures occur. On the side of the institution, E-learning courses have also some advantages like space economy and a lower resource occupation. Although this physical economy, E-learning demands more resources; preparing lectures and managing student records consumes time and human resources.

Here is where the so called Learning Management Systems (LMS) appeared. Basically a LMS provides functionalities to manage student records, to facilitate communication between students and between students and teacher, to control accesses and produce statistics, schedules, evaluation and an open platform to help teachers make lecture content available online. However they do not dictate what kind of technology or format should be used to prepare those contents. Although this issue can be seen as an advantage in certain contexts it leads to a format anarchy and makes support for content production impossible.

ADRIAN is being created in a special context: the authors are computer science teachers, part of them belong to the campus E-learning task force and they have the responsibility of several courses that should be offered in E-learning format. Each component of ADRIAN started as an individual project. These

projects gave birth to a set of prototypes that are now being further developed for real cases use. In parallel we were developing another project called METAMORPHOSIS [clei03,coopmedia03] aiming at exposing and integrating information systems on the web through the use of ontologies to specify the different views. When we joined the set of prototypes we have developed for E-learning content production with the help of METAMORPHOSIS, ADRIAN was born.

For the moment, ADRIAN is composed by four modules to assist the creation of: tests and exams; guided lessons and guided lab sessions; multimedia presentations; and an interface to glue all the others. In the following sections we start by explaining the integration of the three content production modules. Then we will describe each of the individual modules.

2 ADRIAN architecture

Figure 1 illustrates ADRIAN's operational architecture.

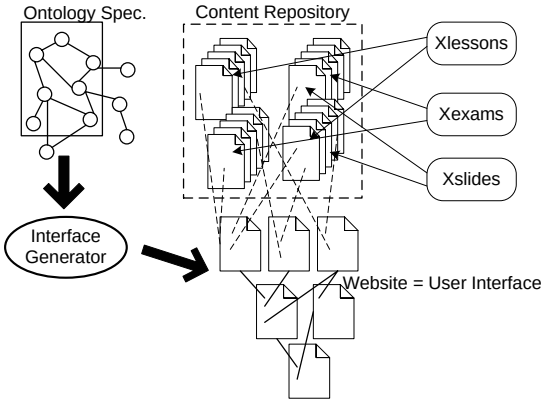

Figure 1: ADRIAN architecture

In the architecture described by figure 1 we can distinguish four major parts:

1. **Content Provider Applications** – Currently this part comprehends three applications:
   - **Xlessons –** to produce lectures: lecture notes, laboratorial guided sessions, exercise sheets, …
   - **Xexams –** to assist teachers in the production of tests and exams: multiple choice, true and false, development, progressive, …
   - **Xslides –** to assist the production of slide based presentations.
2. **Content Repository –** Normally it will be a subtree in the server filesystem. At present this structure should be *frozen* and *known* to the other applications. An ongoing project will take care of these restrictions enabling the user to dynamically change the Content Repository structure. The structure we have defined and that we are using is presented in figure 2.
3. **Ontology –** This part corresponds to an abstract specification of the Content Repository structure. The content provider still has to specify this by hand. There is an ongoing project to compute most of the ontology automatically. The idea is to crawl in the Content Repository structure collecting superclass-subclass relationships and the list of all documents produced so far and to generate the corresponding ontology. At the end the user will be able to add new relationships to the ontology. This ontology represents the input to the Interface Generator, a tool that we have developed that generates a complete website to access and navigate among a set of resources described through an ontology specification.

4. **Interface –** This part corresponds to a website from where the user can access all the produced documents.

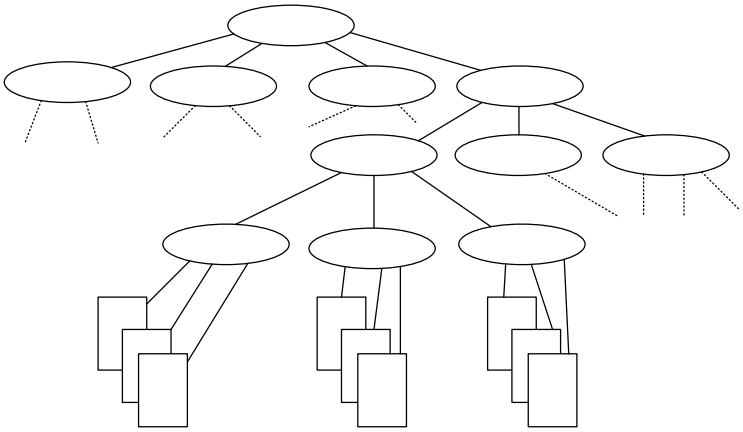

Figure 2: Content Repository structure

In the following sections we describe with more detail each of these components.

3 Content Provider Applications

Our content provider applications share a common philosophy: they are based on the principles of structured documentation and they use descriptive markup to structure the documents that are produced. With this statement we mean that the format of the content being produced is not free. It has a textual representation and has its structure completely described through the use of markup and formally specified with a grammar.

The advantages of descriptive markup are well documented in the bibliography published so far [???] but we can enumerate the most important ones:

- Portability – since the markup is descriptive it has no operational meaning; it is possible to move documents between different systems and application without any changes.
- Flexible formatting – descriptive markup leads to a complete separation between content and form; in order to visualize a document one has to associate it with visual form specification; later on if one wants to change the look of it information, only needs to change this specification the content remains unchangeable.
- Longevity – since there are no ties with operational systems and software platforms it is easy to use and reuse this information in tomorrow systems.

In order to clarify this technological choice, the following subsection explains some inherent to the descriptive markup paradigm.

3.1 Descriptive Markup

The idea of using descriptive markup in an electronic publishing environment dates back to the 1960's. However it was only in 1986 that SGML [???] has emerged as an ISO standard (ISO 8879). SGML is a meta-language with which is possible to define specific markup languages. Although it has some advantages its complexity led to a small community of users. To overcome this problem XML [???] was presented to the public in 1998. XML is a lot lighter, easy to process and new tools and environments appear everyday.

An XML document is a logic structure, a hierarchy of components. Each component can be differentiated from the others through the use of markup that is added to the document. According to this perspective a document has two types of information: data and markup. The following example shows a piece of a document that specifies a lab guided session.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<AulaPrática>
  <meta>
    <disciplina>Processamento Estruturado de Documentos</disciplina>
    <data>2003.10.01</data>
    <objectivos>
      <para>O objectivo principal desta ficha é familiarizar o aluno
com o XPath.</para>
      <para>Para ...</para>
    </objectivos>
    <recursos>
      ...
    </recursos>
  </meta>
  <corpo>
    <introdução>
      <para>Para, ...
      </para>
      ...
    </intridução>
    ...
  </corpo>
</AulaPrática>
```

This example only presents two components of an XML document. An XML document has three possible components:

- **XML declaration** – All XML documents must begin with this declaration:

```
<?xml version=”1.0” encoding=”ISO-8859-1”?>
```

- **DTD** (Document Type Definition) or **Schema** – A grammar that specifies what markup is required, what is optional, where it is required and where is optional. This grammar defines the markup language, in other words, a DTD or Schema defines an XML dialect.
- **The document –** This component corresponds to the document itself. It is composed by text, markup, and optionally a reference to a DTD or Schema.

An XML document that follows a DTD or Schema is said to be a valid document according to that DTD or Schema. An XML document that does not follow any DTD or Schema is said to be a well formed document.

Figure 3 gives an idea of XML documents lifecycle. This figure illustrates the methodology followed to develop the three applications being discussed in this section.

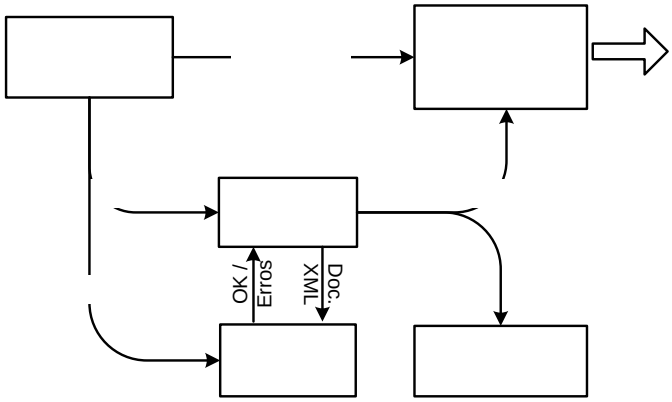

Figure 3: XML documents lifecycle

Figure 3 illustrates the structured documents lifecycle with 5 stages: analisys, edition, validation, storage and formatting. The analisys stage corresponds to the study of a kind of documents. This stage is expected to produce a DTD or a Schema that completely defines the structure for a certain class of documents. After there is a small cycle between two stages: the user edits a document and asks the editor to test its conformance with the DTD; if there are reported errors the user will correct them until the document passes the validity check. This stage (2 stages: edition+validation) will produce valid XML documents. With an XML valid document one can store it or process it in order to obtain a specific output. There are many solutions for storage that will not be discussed here. The transformation process is normally specified in an XSL stylesheet [???] (XSL is an XML syntax that is used to specify transformation processes in a declarative/functional way).

Analisys (DEs, ELM tree)

Let´s see how these concepts were applied to the three content provider applications.

3.2 Tests and Exams

This is probably the most complex type of document in the ADRIAN framework. In an E-Learning context we normally refer to online exams. There are two species of online exams: static (the exam is just displayed inside an ordinary web page (this situation is common to most of other types of documents) and dynamic (the exam is displayed but students are expected to interact with it, they can solve the questions and have immediate feedback from the system). ADRIAN supports dynamic exams and that is the kind of exams that is presented in the following.

DTD or Schema

Dynamic exams can be divided in the following types:

- **Multiple choice** – each question has a set of answers; the student must pick the wright one.
- **True and False** - each question has a set of answers; for each answer the student must give a true or false answer.
- **Development** – the student has a blank area to develop his answer to the question.

DTD or Schema

The ADRIAN tests and exams application is being developed to support all these three kinds of exams.

Further more, functionally we can classify exams in one of the following types:

- **One attempt** – the student has only one attempt to solve the exam (this is the traditional approach).
- **Time limit** – the student has to solve the exam inside a specific time interval.

- **Progressive** – the exam is presented by levels; to access the following level the student has to reach a minimum in the current level.
- **Random** – the system scrambles the questions; the order in which questions are presented is always different.

An exam does not need to strictly belong to one of those types. It can be a mixed of all or some of those types. In order to achieve this we deal individually with each question. Questions are the building blocks of our exams.

3.3 Slide based Presentations

3.4 Lessons and Laboratory Guided Sessions

4 Content Repository

5 Ontology

6 Interface

7 Conclusions

References

1. Eve Maler and Jeanne Andaloussi, Developing SGML DTDs: From Text to Model to Markup. Prentice-Hall, 1996.
2. Eric Van Herwijnen, Practical SGML. Kluwer Academia Publishers, 1994.
3. Elliotte Rusty Harold and W. Scout Jeans, XML in a Nutshell. O'Reilly, 2001.
4. Charles Goldfarb, The SGML Handbook. Clarendon Press – Oxford, 1990.
5. Philippe Martin and Laurence Alpay, Conceptual Structures and Structured Documents. *INRIA*, *ACACIA* Project,1998.
6. David Megginson, Structuring XML Documents. *Prentice*-Hall, 1998.