SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2007; **37**:441–461
Published online 2 November 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.779

A highly modular and extensible architecture for an integrated IMS-based authoring system: the <e-Aula> experience

José Luis Sierra[1], Pablo Moreno-Ger[1], Iván Martínez-Ortiz[2] and Baltasar Fernández-Manjón[1,*,†]

[1]*Dpto. Ingeniería del Software e Inteligencia Artificial, Fac. Informática, Universidad Complutense, 28040 Madrid, Spain*
[2]*Centro de Estudios Superiores Felipe II, Aranjuez, Spain*

SUMMARY

The <e-Aula> platform is a new experimental e-learning environment that adheres closely to IMS Global Learning Consortium, Inc. e-learning standards in order to facilitate their applicability in different learning scenarios. <e-Aula> is equipped with an integrated modular and extensible architecture for the authoring of IMS-compliant learning materials focused on the IMS *manifest*. This *manifest-driven* architecture facilitates maintenance and promotes the evolution of the authoring system in <e-Aula>, both of which are mandatory requirements in the successful production and maintenance of content for many different specialized learning domains. In this paper, we describe this authoring system, its manifest-driven architecture and its implementation using well-known and robust Java-based Web technologies. Copyright © 2006 John Wiley & Sons, Ltd.

Received 1 August 2005; Revised 14 April 2006; Accepted 28 June 2006

KEY WORDS: E-learning; IMS-based experimental platform; IMS-based authoring system; <e-Aula>; J2EE; Struts

1. INTRODUCTION

E-learning is a broad and very active field in which any type of learning process aided by information and communication technologies can be included. A large number of universities have decided

*Correspondence to: Baltasar Fernández-Manjón, Dpto. Ingeniería del Software e Inteligencia Artificial, Fac. Informática, Universidad Complutense de Madrid, C/ Profesor José García Santesmases, s/n, 28040 Madrid, Spain.
†E-mail: balta@fdi.ucm.es

Contract/grant sponsor: The Spanish Committee of Science and Technology; contract/grant numbers: TIC2002-04067-C03-02, TIN2004-08367-C02-02 and TIN2005-08788-C04-01

to introduce E-learning facilities by using e-learning platforms. These platforms enhance instruction by allowing teachers to manage learner data, offering personalized learning material, and tracking learner activity inside the environment [1]. Nevertheless, such approaches are frequently exposed to specific and recurring obstacles that are clearly identified but not always carefully confronted [2,3]. One of these obstacles is the obsolescence of the material. It is common to see that excellent material, obtained at an enormous cost by employing multidisciplinary groups of experts, quickly becomes obsolete upon the appearance of new technology on the market (e.g. the transition from videodisc or CD-ROM to a Web-based environment). Another recurrent obstacle is the lack of support for reuse. While teachers will often use material already prepared for other subjects to their advantage (e.g. figures or examples proposed in books and even material borrowed from other teachers), the platform often does not provide them with specific facilities to aid this reuse of the contents in the creation of new courses. These obstacles can be adequately addressed with the use of e-learning standards.

Several initiatives on recommendations and standards covering the needs of e-learning environments that allow a high durability and reusability of the contents have been proposed (e.g. IMS [4], ADL-SCORM [5], AICC [6] and IEEE Learning Technology Standards Committee (LTSC) [7]). Among these initiatives, we have focused our attention on the proposals of the IMS Global Learning Consortium, Inc., since they are the most comprehensive collection of specifications and are becoming the most solid basis for industrial standards in the field. IMS is an international consortium which brings together experts from the different fields involved (directly or indirectly) in e-learning technologies. The main concern of IMS is the interoperability between e-learning systems, and for this purpose the consortium has delivered several specifications covering different aspects of the interoperability process. The core of the IMS specifications is IMS Content Packaging (IMS CP) [8], which defines how to aggregate educational contents into *packages* in order to allow the interchange of such contents between different heterogeneous systems. Other relevant IMS proposals include the following: IMS Question and Test Interoperability (IMS QTI) for tests and assessments [9], IMS Learner Information Package (IMS LIP) for storing and interchanging information about learners [10], IMS Simple Sequencing (IMS SS) [11] for ordering the learning content in a course and IMS Learning Design (IMS LD) [12,13] for describing the learning activities and the interaction between learners, tutors and the platform.

Since they are geared more toward characterizing the different aspects of the instructional processes, IMS specifications do not constrain the types of learning contents finally included. Indeed, these contents will be highly dependent on each particular learning scenario. While general-purpose and widely used types of contents (e.g. HTML pages, PDF documents and media in several formats) can be produced using widely-available edition and authoring tools, more specific types of contents (e.g. a very specialized documental language used by linguists during the production of educational material, as seen in [14]) may require domain-specific authoring support. In the same way, while the delivery of passive content may rely on standard client-side facilities (e.g. plug-ins for audio, video or other media), more interactive content (e.g. educational videogames or simulation tools) may rely on a smart and content-specific distribution of functionalities between the client and the server. To address the domain-specific nature of the production, maintenance and deployment of learning contents in concrete learning scenarios we have implemented <e-Aula> [15–17], an experimental e-learning platform supporting several IMS specifications. The <e-Aula> project began in 1999 as a platform to evaluate the applicability of e-learning standards and to obtain first-hand experience with several e-learning specifications, which at this time were not established and were frequently changing.

DOI: 10.1002/spe

Currently, <e-Aula> has grown far beyond its initial evaluation purposes, evolving into a fully featured e-learning solution that bases its power on a close adherence to established standardization proposals, and is equipped with an easy-to-use and easy-to-extend built-in authoring system. This system allows for the edition and deployment of a wide and open range of learning resources. In addition, it has been designed to be easily specialized on different application profiles, and includes flexible and robust importation/exportation facilities rooted in its adherence to IMS standards.

The authoring system in <e-Aula> is organized according to the so-called *manifest-driven* architecture, which constitutes its main novelty. According to this architecture, the entire system is conceived as a coordinated set of processors for the XML domain-specific markup language used to structure the IMS *manifests*, the XML documents that describe the structure of the IMS content packages in the IMS CP specification. This organization has yielded a number of benefits for the implementation of a highly modular and integrated system that adheres to the specifications but is resilient to their evolution and flexible enough to cover the broadly open aspects of IMS specifications such as content types or delivery methods. In addition, the built-in authoring support lessens the burden posed on authors by the use of several disaggregated third-party authoring tools, which is especially critical in a continuously evolving scenario. Finally, the system fully addresses the open nature of IMS specifications and therefore can easily be specialized on different domain-specific learning scenarios and experiences, even on those supporting highly interactive contents. In this paper, we describe this architecture and its implementation using Java technologies.

The structure of the paper is as follows. Section 2 provides the background to IMS CP which is required to understand the rest of the paper. Section 3 gives a brief description of the authoring system in <e-Aula> and describes its manifest-driven architecture. Section 4 summarizes the details concerning its implementation. Section 5 discusses some related work. Finally, Section 6 gives several conclusions and outlines some possible areas for future work.

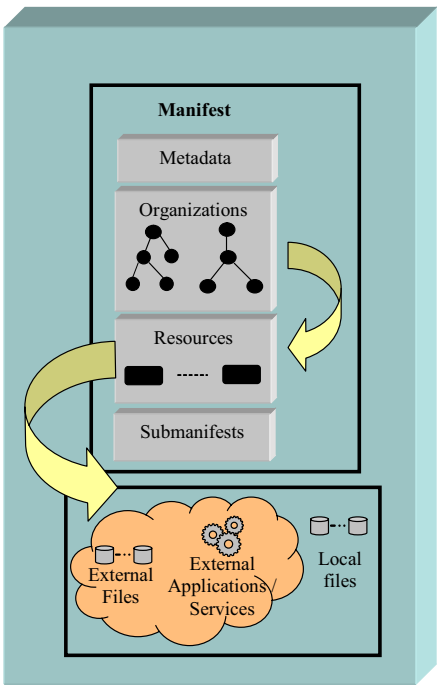
2. THE IMS CONTENT PACKAGING SPECIFICATION

As stated before, the IMS CP specification is one of the bases of IMS specifications, since it establishes how to package educational contents together for purposes of interoperability. In addition, this specification also serves as a packaging mechanism for other IMS specifications, either using the IMS CP extension points (such as IMS LD) or simply referencing the documents that contain the information (such as IMS QTI). These packages are usually distributed as zipped files, and their structure is depicted in Figure 1(a). In this way, a package is formed by a collection of learning contents and a *manifest*. As also indicated, this manifest is an XML document [18] that reflects global information about the package, the structure of the contents, their types and their possible organizations. More precisely, the manifest contains the following elements.

- A section of metadata summarizing global (*meta*)*information* about the package. This metainformation usually follows the Learning Object Metadata (LOM) specification defined by the IEEE LTSC [19].
- The description of package *resources*. Resources are the basic content units of the package. As seen in Figure 1, typical resources are made up of a set of *files*: a main file and a (possible empty) set of secondary files (e.g. a main HTML file and the images referred from this file).

DOI: 10.1002/spe

(a) (b)




```
<manifest identifier="Photoshop Package">
  <metadata> (…) </metadata>
  <organizations default="ORG1">
    <organization identifier="ORG1">
      <title>Photoshop Tutorial</title> (…)
      <item identifier ="ORG1_3">
        <title>Module 1 - Basics</title>
        <item identifier="ORG1_3_0"
              identifieref="rec3.0">
          <title>Interface</title>
        </item>
        <item identifier ="ORG1_3_1"
              identifieref="rec3.1">
          <title>Toolbox</title>
        </item>
        <item identifier ="ORG1_3_2"
              identifieref="rec3.2">
          <title>Palettes</title>
        </item>
      </item (…)>(…)
    </organization>
  </organizations>
  <resources> (…)
    <resource identifier="res3.0" type="webcontent">
      <file href="Photoshop3.0.html"/>
    </resource>
    <resource identifier="res3.1" type="webcontent">
      <file href="Photoshop3.1.html"/>
      <file href="fig3.1.1.gif"/>
      <file href="fig3.1.2.gif"/>
    </resource> (…)
  </resources>
</manifest>
```

Figure 1. (a) Structure of an IMS package; (b) fragment of an IMS manifest extracted from a course deployed in <e-Aula>. These figures have been simplified for presentational purposes and for the sake of clarity.

In turn, these files can be associated with local archives with learning content as particular cases. However, the flexible nature of IMS specifications does not limit resources and files to being local. Resources can also be external assets referenced to using an URL. Therefore, any service or tool included in the learning process defined by the package (e.g. a Web application and/or service) can become a resource. This also holds for individual files inside a resource, which can correspond with external objects on the Internet using absolute URLs. Moreover, the resource can include metadata about itself and define the type of content within it. Finally, each resource must have a unique identifier.

- The *organizations* of the resources. Each organization represents a tree structure whose nodes can refer to resources. The nodes in this tree are called *items* and contain a reference to a corresponding resource using the unique identifiers of these resources. Therefore, an organization provides a tree-based structure of the resources of the package (and thus, of its leaning content). Also note that a manifest can include several organizations, each providing an alternative way to organize the contents and therefore a different view of the package.
- The *submanifests*. A manifest can contain other, simpler manifests that in turn exhibit the same structure outlined here.

DOI: 10.1002/spe

Figure 1(b) depicts a manifest adapted from an ADL-SCORM sample course about Photoshop [20], which has been deployed in <e-Aula>. This example shows a tree-based organization linked to different resources by means of the usual XML *id-idref* mechanism [18]. In turn, the resources contain URLs pointing to the actual content.

As a final remark, it is important to note that IMS does not impose any restrictions on the format or type of content files, nor on the metadata schema to be used. These aspects are established in each particular *application profile*. According to IMS, an application profile is the customization of a specification to meet the needs of particular communities of implementers with common application requirements. For instance, in <e-Aula> we have defined the *eAulaAP* application profile, which includes resources such as a cover, a glossary and a list of FAQs whose contents are documents marked up with XML-based domain-specific markup languages [21–24]. This possibility adds all the benefits of the content-structuring power of descriptive markup to the IMS structuring proposal [25].

Despite the proposal and development of the *eAulaAP*, one of the highlights of the architecture described in this work is its potential modular evolution to support different application profiles and thus accommodate the IMS CP open nature. In the following sections, the organization of the system is described with special attention to the relations between design decisions and IMS specifications.

3. THE AUTHORING SYSTEM IN <e-Aula> AND ITS MANIFEST-DRIVEN ARCHITECTURE

The <e-Aula> platform is equipped with a flexible authoring system enabling the edition, preview, importation and exportation of educational contents. The system allows authors to maintain a *repository* of IMS-aware learning material. For this purpose, it offers the following four main functionalities.

- *Edition*. This functionality enables the authoring of the different aspects of an IMS package. On the one hand, the global structure of the package can be edited. On the other hand, users can edit the organizations by adding and removing items, as well as by assigning resources to these items (Figure 2(a)). Finally, they can edit the actual contents associated to the resources by using suitable editors for each content type (Figure 2(b)).
- *Presentation*. This functionality allows authors to preview IMS packages. Authors start by choosing an organization (if the package contains more than one). Then the organization chosen is displayed as a tree structure (Figure 3(a)). Authors may navigate this structure and visualize the contents associated with the resources (Figure 3(b)).
- *Importation*. This facility allows the incorporation of off-the-shelf IMS packages produced in other IMS-aware systems. Each incoming package is examined for unsupported features and, if required, a list of adaptation tasks is produced. Some of these tasks may require further interaction with the user, and other tasks can create new tasks that are added to the adaptation list. Therefore, importation in <e-Aula> is a complex and highly dynamic process that ensures the quality of the e-learning material incorporated and its suitability for each specific learning scenario.
- *Exportation*. This facility makes the educational resources produced in the system available for others in the form of IMS packages.

DOI: 10.1002/spe

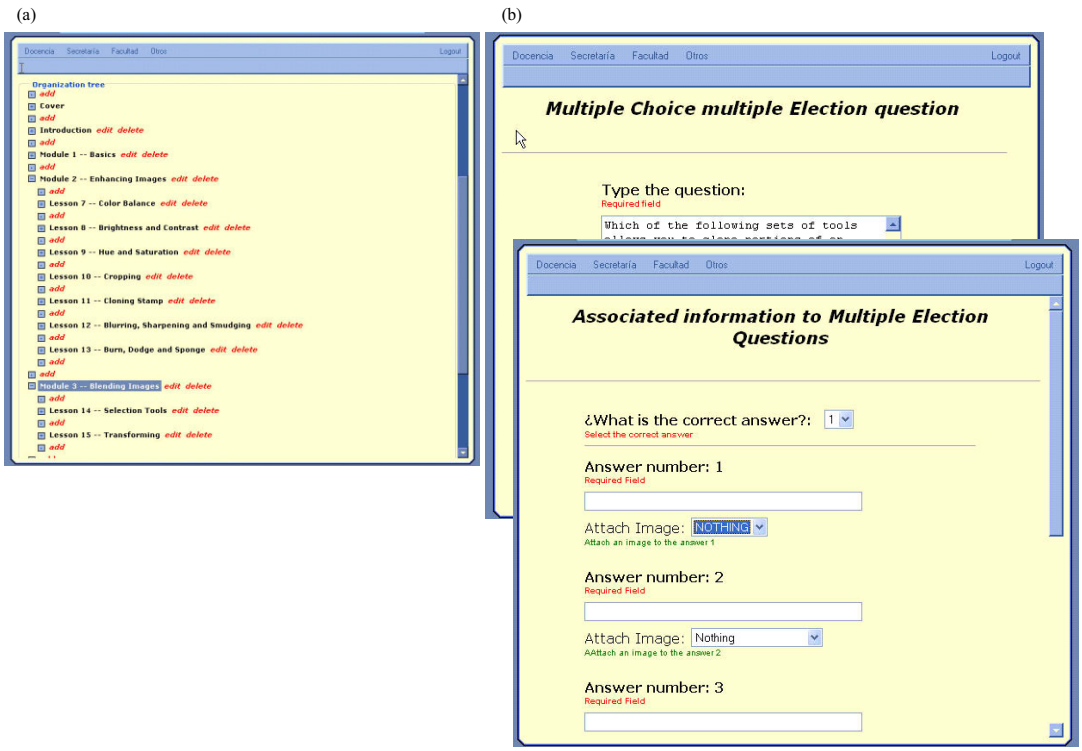


Figure 2. (a) Edition of a new organization (notice the edition style of a tree-like structure); (b) edition of a resource content (a QTI test) with a resource-specific editor for QTI.

This authoring system has been designed to be highly modular in order to accommodate our research and development needs and accommodate the open nature of IMS specifications. Therefore, new editors and players, as well as new adaptation tasks and strategies, can easily be integrated, enabling the specialization of the system in each domain-specific application profile. To achieve this we have used a *manifest-driven approach* according to which *the manifest is used as the key element for driving the design and architecture of the entire system beyond the usual interoperability processes*. This architecture is sketched in Figure 4. According to this approach:

- The repository stores IMS-aware representations of the learning materials. This means that a representation of the IMS manifest for each package is explicitly maintained. Therefore, all the operations performed on this material will be reflected in these representations.
- Edition and presentation are architected following a *delivery approach*, with *resource processors* able to select the most suitable *resource editors* or *players* for each type of resource. Hence each resource processor enables a specialization of the system for each particular application profile.

DOI: 10.1002/spe

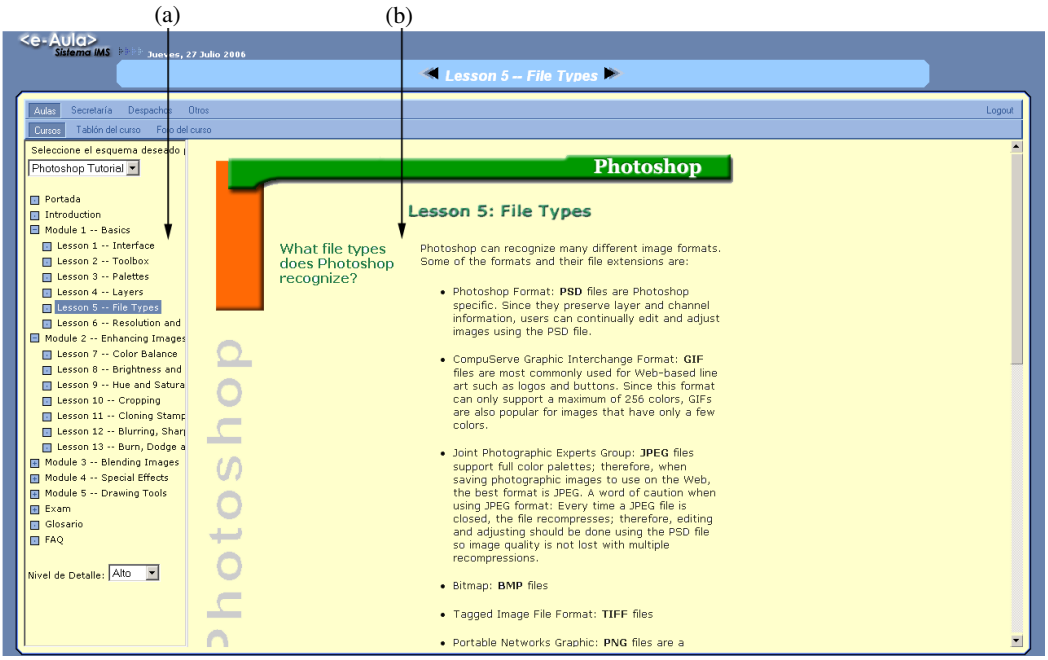


Figure 3. (a) Presentation of an organization; (b) presentation of a resource with a corresponding resource-specific player.

- Importation is architected using an *agenda-based approach*, where an agenda of *adaptation tasks* is used to cope with the complexities of this operation in a dynamic and modular way.
- Finally, exportation is straightforward due to the explicit representation of the manifests in the repository.

The next sections describe this architecture.

3.1. The repository

The repository within the architecture acts as a storage layer for the IMS packages included in the system. All subsystems are organized around this entity, which offers the functionalities invoked by the rest of the application for operations on full packages (e.g. loading, updating and deleting existing packages, creating new packages, etc.) as well as on content files (e.g. loading and saving files, storing new files, etc.). The presence of this layer isolates the different subsystems from the low-level details of course storage.

One of the main responsibilities of the repository is to offer a common and coherent model of all the learning material to the other modules. In particular, the repository takes care to ensure coherence in the

DOI: 10.1002/spe

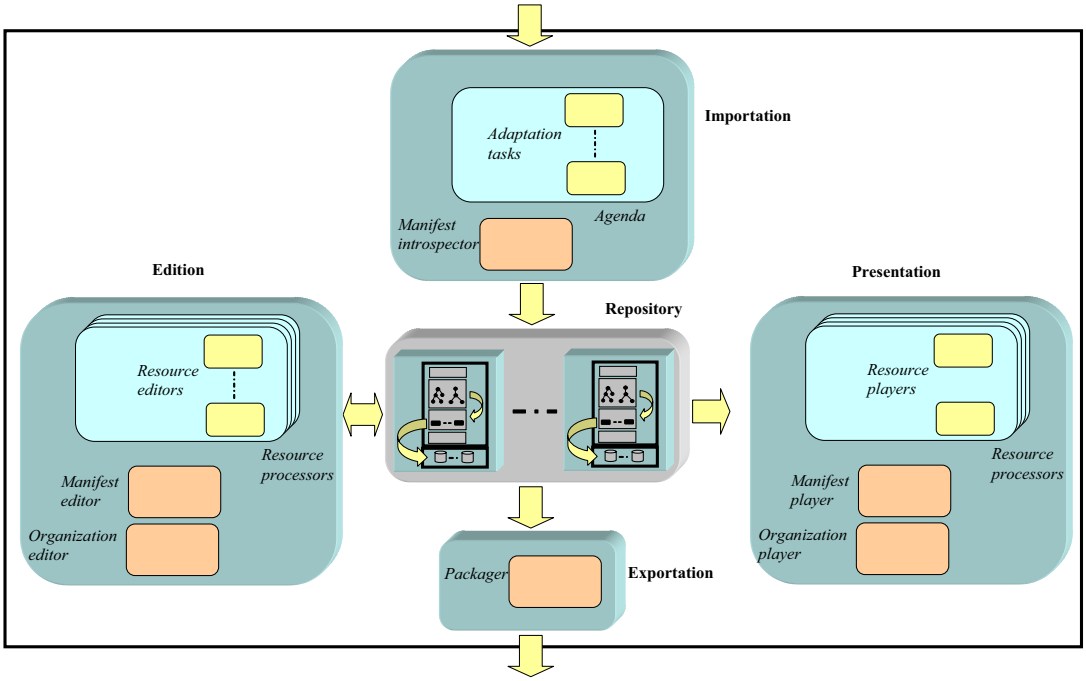


Figure 4. Architecture of the authoring system in <e-Aula>.

concurrent access to this material. In addition, the repository must offer efficient access to the packages stored. For this purpose, several caching techniques can be applied (e.g. storing of IMS packages in an unzipped format, and keeping an in-memory representation of the manifest file using an entity class) and advanced database support can be used.

3.2. The edition module

The edition module is based on a *delivery* approach enabling it to react to different application profiles and different content types and process them accordingly (see Figure 5).

- The module maintains a table that associates a *resource processor* with each application profile supported by the authoring system. Therefore, when an existing package is loaded or a new package is created, its profile is used to obtain the adequate processor for handling all the requests related to the edition of the package's resources. For existing packages this profile is obtained from the package's manifest, while for new packages it is requested from the user.
- In turn, each resource processor maintains a table relating each content type to an appropriate *resource editor*, which is a component capable of editing content of this type. Therefore,

DOI: 10.1002/spe

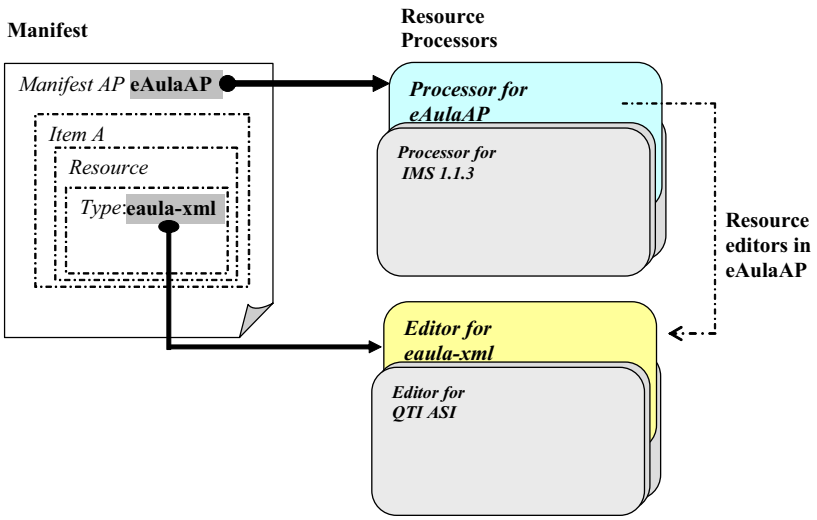


Figure 5. The delivery strategy of the edition module exemplified. The resource processor for the *eAulaAP* application profile is loaded according to the corresponding profile found in the manifest. The resource editor for the content type *eaula-xml* is loaded according to the type of resource.

when the user orders the edition of a resource, the manifest is used to obtain the type of the resource and this type is used by the active resource processor to determine the editor to be launched. Notice that the way of managing the different kind of files included in the resource is editor-dependent. For instance, an editor can decide to make local copies of the external files of a given type. Another editor can only allow the change of the URLs for such files (referencing to package internal files or external contents/services). A third editor can collaborate with a third-party application accessible by using a Web service in order to update any appropriate pieces of external learning content.

- In addition to this delivery strategy, the module uses a *manifest editor* component to edit the global structure of the package and an *organization editor* to let the user edit the package's organizations.

This structure allows the incorporation of new resource processors with new resource editors in a clean, modular way without interfering with the overall editing behaviour of the authoring system, as we have found during the development of <e-Aula>. As an illustrative example, we could cite the integration of support for IMS QTI in the authoring system. An independent team from the <e-Aula> project developed a QTI Lite engine in the form of a standalone Web application. The integration was performed by implementing a resource editor for QTI by wrapping the edition facilities of the independent application (see also Figure 2).

DOI: 10.1002/spe

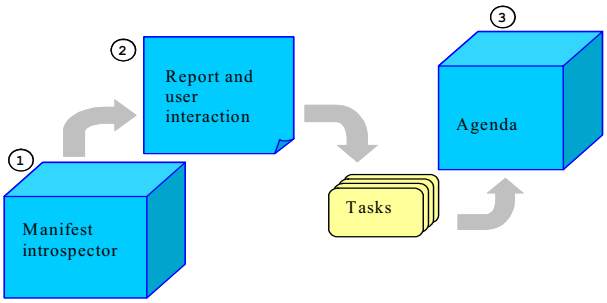


Figure 6. The steps of the importation process. The execution of some tasks may include interaction with the user. Task execution and user decisions may add new tasks to the agenda.

3.3. The presentation module

The presentation module is organized in a similar way to that of the edition module.

- The content presentation environment reuses the ideas used in the edition module to trigger the appropriate *resource players*. Notice that for interactive contents, these players will address not only static presentation but may also perform other tasks, such as tracking the interaction with the learner. Also notice that the way to play the learning content (including external content) is entirely player-dependent. As with edition this approach allows a great flexibility, making it even possible to involve third-party remote applications accessible through the Internet.
- Navigation throughout the structure of the course is facilitated by an *organization player* component. This component is analogous to the organization editor in the edition module, but this time the tree-like representation of the manifest is not editable. In addition, there is also a *manifest player* component, which is used to present the overall structure of the package to the user, letting them choose the organization to be used during navigation.

As with the edition module, the architecture of the presentation module allows the incorporation of new presentation facilities in a straightforward manner. This feature has proved to be very useful during our experimentation with <e-Aula>, letting us integrate, for instance, an interactive player based on the QTI system mentioned above.

3.4. The importation module

The richness and open nature of the IMS CP specification makes it so flexible that, for example, packages can contain a variety of content types unknown to any importation system or content files can be based on different versions of the specification. The architecture of the importation module must be flexible enough to cope with the complexities of the importation process. It must implement a flexible behaviour that is capable of reacting when confronted with different problems, even with

DOI: 10.1002/spe

the possibility of querying the user when more information is needed to perform the importation. Because of this, we propose an implementation based on an *agenda* similar to that proposed in [26] to simulate discrete systems.

According to agenda-based organization, when a package is imported, the system parses the manifest, adding new tasks to the agenda to solve the troubles encountered during the scan. These tasks (especially those that involve a query to the user) can create other tasks and add them to the agenda if needed for their resolution. In this manner, a complex process that requires a heterogeneous set of actions is dynamically split into simple tasks (see Figure 6).

- The importation begins with a deep scan of the manifest. This is carried out using a component called the *manifest introspector*. This generates a report that profiles, among other things, whether the manifest is a well-formed XML document, which version of the standard it follows, which other schemas (if any) are needed to understand it and under what application profile it has been developed. This report is presented to the user, who then decides how to continue.
- Guided by the user's response, the system may generate a list of initial tasks for the agenda and start working on them. Such tasks may include the modification of the manifest file, modification of the application profile, adaptation of some resources or physical installation of the package.

This agenda-based implementation makes it possible to add new types of adaptation tasks during importation in a transparent way (e.g. to support a new content type). Such an organization dramatically enhances the modularity and maintainability of this complex subsystem. A representative and realistic example of this feature at work is our experience in adapting incoming packages to our *eAulaAP* application profile. All that was needed to adapt general IMS CP-compliant packages to this profile was to include the required cover, glossary and FAQ resources during the importation process. This was accomplished by implementing adaptation tasks that generate such files from a default template and insert references to them in the incoming manifests.

3.5. The exportation module

In systems that use IMS CP as a basic interoperability format, but whose internal architecture is not necessarily driven by this specification (see Section 5), the process of exporting content following the proposed standards can be a complex task. It might imply scanning the course, finding out its structure and type, packaging the contents and creating a valid manifest to be delivered with the package. For instance, in a system where the structure of the learning contents is internally stored in a relational database following a domain-specific optimized information model, the packaging process will imply an overall sequence of queries to such a database in order to rebuild the IMS manifest. However, in the manifest-driven architecture this process is extremely simple, and can be carried out by a simple packager component, which produces a zipped IMS package from the information maintained in the repository.

4. IMPLEMENTING THE MANIFEST-DRIVEN ARCHITECTURE

The previous section introduced the conceptual view of manifest-driven architecture. This section describes how those general concepts can be modelled as a Web-based application with special attention to modularity and extensibility requirements.

DOI: 10.1002/spe

Since the possibility of distributing the system under an open source license has been considered, all the implementation technologies must be licensed as open source as well. This restriction naturally led to the adoption of Java technologies given the spectrum of open source initiatives surrounding this platform. For this reason we have based our implementation in <e-Aula> on Sun Microsystems' J2EE platform [27] complemented by the Apache Foundation's Struts framework [28]:

- From J2EE we have adopted the multi-tier organization according to which applications are layered in different tiers (*client*, *Web*, *business* and *persistence*).
- From Struts, we have adopted the organization in terms of the classical Model-View-Controller (MVC) design pattern [29]. In the MVC design pattern, a central *controller* manages program flow. The *controller* receives requests and acts on the *model*, which represents the system's information and state. Control is then forwarded back through the controller to the appropriate *view*, which manages the information display. The use of the MVC pattern clearly decouples the view and the model, contributing to making applications significantly easier to create and maintain. In Struts, controllers can be characterized as set of *actions* (i.e. components encapsulating the response to a user's interaction) that act as façades to an arbitrarily complex *domain logic*. In addition, we use client-side scripted HTML pages as well as Java Server Pages (JSPs) [30] to configure the views. Finally, models can be characterized by a *data management* logic connected to a persistent storage (database) support.

Both platforms suggest how the implementation should be organized and, although not obvious at first sight, both approaches can be blended in an elegant manner. With this, the <e-Aula> implementation of the manifest-driven architecture is disposed on a layered organization that takes full advantage of the power offered by the MVC pattern. Notice that each main module (edition, presentation, importation and exportation) has its own view and controller, which interact with a common model represented by the repository and the manifests within it.

The next sections address the main implementation details of each module in the architecture from the simplest to the most complex.

4.1. Implementing the repository

The repository is the core of the system and represents the common model shared by the remaining modules. As previously mentioned, the repository controls the access to all the learning material and gives the rest of the modules a coherent view of this material.

The structure of the repository in the authoring system is depicted in Figure 7. As suggested in this figure, IMS packages are deployed physically in an unzipped format by using a directory of the file system for each package whose name is the unique identifier included in package's manifest. It is important to note that all the directories are situated at the same level (i.e. the resulting structure of directories associated with the unzipped packages is flat). This means that sub-packages inside a composite package are allocated in their own directories. The rationality of this design decision is to allow these packages to be edited and displayed as independent packages in a uniform way. The repository will take care of the dependencies by avoiding, for instance, the removal of a package if it is used as a sub-package by another package. Moreover, sharing sub-packages is not allowed by the repository. For this purpose a copy must be made and used instead.

DOI: 10.1002/spe

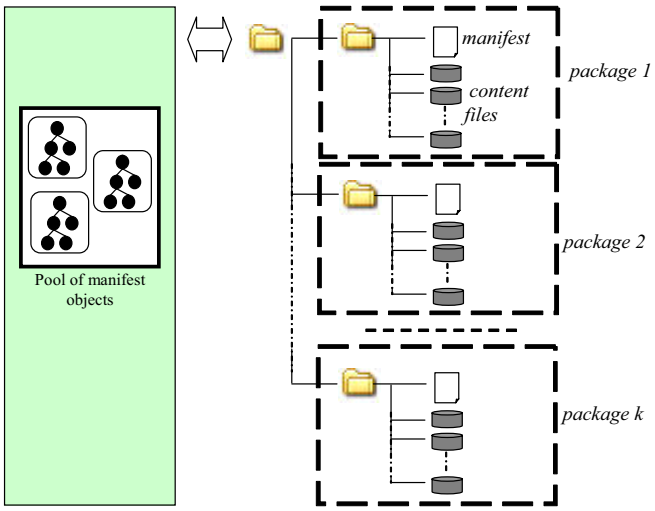


Figure 7. Structure of the repository.

The repository ensures consistent access to these directories (e.g. while two concurrent presentations of the same package are allowed, the repository prohibits two or more simultaneous editions). In addition, and for reasons of efficiency, the repository maintains an in-memory *pool of manifest objects*. The first time that a package is accessed, its manifest is parsed and a *manifest object*, containing a document object model representation [31] of the tree, is added to this pool. All subsequent accesses will be directed to the object stored. This pool of objects also facilitates coherency maintenance because all the modules gaining access to the same package can share the same manifest object.

Finally, it should be noted that this implementation includes an abstraction layer for the manifest and the operations that can be performed on them. This will allow the substitution of the default filesystem's backend with more sophisticated database support if required.

4.2. Implementing the exportation module

The structure of the exportation module is depicted in Figure 8. This structure, which is the simplest of all the modules as a consequence of the manifest-driven approach, has the following features.

- There is a single *export* action that uses the packager on the directory associated to the package. This action can either succeed (in this case a zipped package will be produced), or fail (this will be the case when trying to export packages currently under edition).
- The view is formed by an *init* JSP, which is the entry point to the module and which produces a page letting the user choose the package to export, and a *result* JSP, which displays a report informing the user of the outcome of the packaging process.

DOI: 10.1002/spe

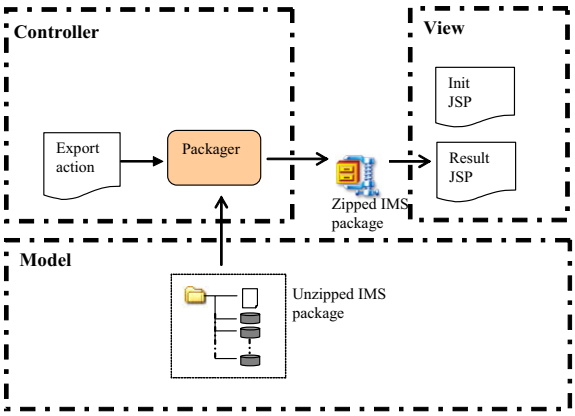


Figure 8. Structure of the exportation module.

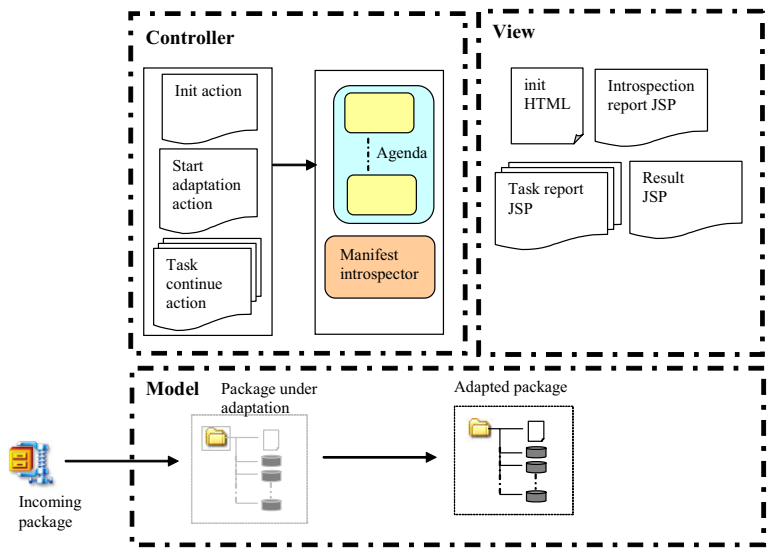


Figure 9. Structure of the importation module.

DOI: 10.1002/spe

4.3. Implementing the importation module

The structure of the importation module is depicted in Figure 9. Although the behaviour of this module can be complex, this complexity is adequately dealt with by an agenda-based architecture as mentioned before. This architecture leads to a reasonably simple implementation.

- The domain logic is composed of the *agenda* and the *manifest introspector*. In turn the agenda is structured as a simple FIFO queue of adaptation tasks. Each task generates a report informing about its execution. If a task fails, the execution of the agenda stops and the incoming package is discarded. If the agenda is emptied, the adapted package is definitively inserted in the repository.
- The entry point is given by an *init* HTML page, which lets the user upload the package to be imported. This page has an *init* action associated that temporally inserts the incoming package in the repository and applies the manifest introspector to its manifest in order to produce the initial report for the user. This report is presented by using an *introspection report* JSP in the view. There is also a *start adaptation* action that interprets the reaction of the user to this report, populates the agenda with an initial set of tasks and activates its execution.
- Reports generated by the tasks are interpreted by associated *task report* JSPs in the view. This kind of JSP, specific to each type of adaptation task, can include controls in the page generated to request further information from the user when required. In turn, the response to these reports is worked out in the corresponding *task continue* action. These actions interpret the reaction of the user to the corresponding reports, adding and/or removing tasks in the agenda when required, and reactivating its execution.
- Finally there is a *result* JSP, which is used to announce the end (success or failure) of the importation process.

4.4. Implementing the presentation module

Figure 10 illustrates the structure of the presentation module. As seen in this figure, the module can be implemented in the J2EE/Struts context in a clean and elegant manner.

- Resource processors are split into a view and a controller part. For each content type supported by a processor there is a *displayer* JSP, which takes care of displaying the resource. In addition, for interactive contents (e.g. an interactive test) there is also an associated *interactor* action in the controller part that takes care of the interaction. Therefore, resource players are implemented as a displayer JSP and for interactive content types by an interactor action.
- The entry point to the module is the *manifest displaying* JSP. The page generated enables the selection of the package to be presented and the organization to be used in this presentation. In response the *display manifest* action is executed, which sets up the suitable resource processor (both in the controller and the view) according to the first step of the delivery strategy described above (i.e. a suitable resource processor is chosen according to the manifest's application profile). Therefore the pair *manifest displaying* JSP–*display manifest* action configures the *manifest player* component in the conceptual architecture.
- In turn, the organization's presentation is produced by the *organization displayer* JSP. It is also interesting to note that the dynamic part of this displayer is configured as an XSLT

DOI: 10.1002/spe

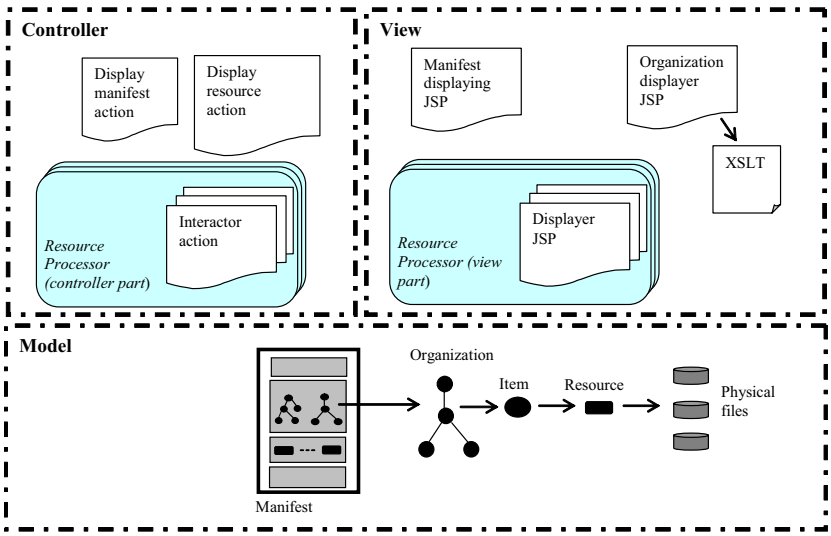


Figure 10. Structure of the presentation module.

transformer [32] which generates the required (HTML) presentation from the manifest document tree. When an item with an associated resource is selected in this presentation, the *display resource* action is executed. Therefore, the *organization player* component is implemented with the pair *organization displayer* JSP—*display resource* action. This last action implements the second step in the delivery strategy: to choose a suitable displayer JSP in the active resource processor. For interactive content types, the interaction will be managed by the corresponding interactor action.

4.5. Implementing the edition module

The edition module is the most complex to implement in the authoring system in <e-Aula> because it must offer remote Web editing capabilities of all package elements. Hence the different editing components introduced in the conceptual architecture (manifest editor, resources manager and resource editors) are split into an action and a view (JSP) part. The main features of this implementation, which is depicted in Figure 11, are as follows.

- The entry point is the *init* JSP, which lets the user choose a package to edit or create a new package. The result of this interaction is worked out in the *init* action which also sets up the appropriate resource processors, both in the controller as well as in the view part.
- The *manifest edition* JSP produces an HTML form that lets the user modify the global structure of the package. The result is processed by the *manifest edition* action. Both assets configure the *manifest editor* component in the conceptual architecture.

DOI: 10.1002/spe

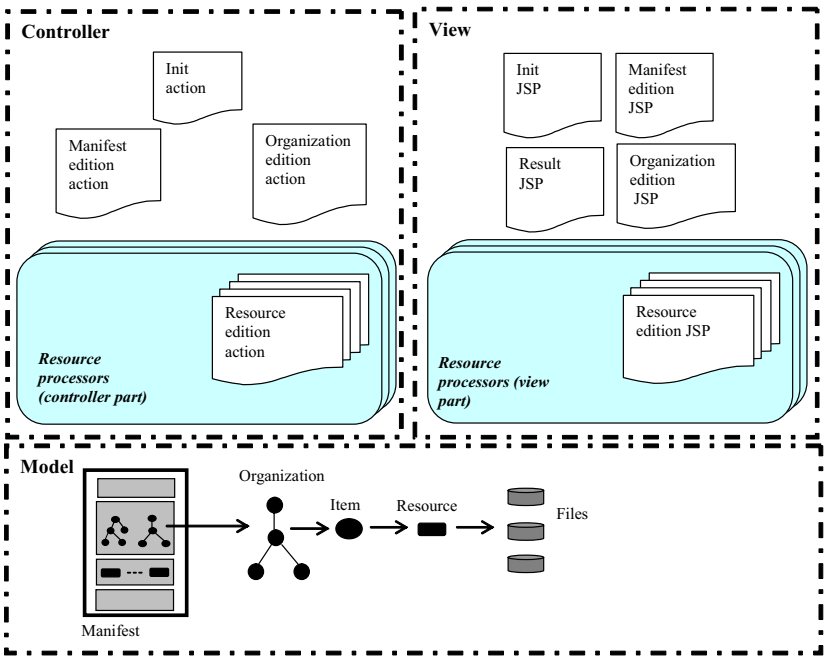


Figure 11. Structure of the edition module.

- The *organization edition* JSP is analogous to the *organization displayer* JSP in the presentation module, but the presentation produced is extended to allow edition (i.e. new controls are included to enable the addition and deletion of new items, as well as their edition). The result of interacting with this presentation is processed by an *organization edition* action that encapsulates the required editing operations. Therefore the pair *organization displayer* JSP—*organization edition* action configures the *organization editor* component introduced in the conceptual architecture of Section 3.
- Finally, each *resource editor* is similarly split into a *resource edition* action and a *resource edition* JSP. The resource edition JSP will produce an HTML page, giving access to the corresponding resource editor. The editing result is processed by the *resource edition* action. Both kinds of assets are grouped into processors oriented to specific profiles, both in the controller and in the view parts. The *init* action performs the first step in the delivery policy, setting up the appropriate processor's controller and view parts on the basis of the manifest's application profile. The *organization edition* action carries out the second step when the user orders the edition of a content file.

DOI: 10.1002/spe

5. RELATED WORK

IMS specifications have been adopted by many e-learning initiatives as a basic interoperability mechanism. They range from commercial e-learning platforms such as WebCT [33], which can be purchased by companies in order to deploy their own content, to initiatives such as ADL-SCORM [5], which provides complementary specifications to obtain high-quality content and systems in a variety of fields. While these initiatives mainly pay attention to IMS standards when it comes to interoperability issues (e.g. content exportation/importation), our manifest-driven proposal goes a step further. Indeed, we propose the use of the specifications (in particular, the IMS manifest) as central mechanisms to conceive, architect, design and implement the system. The manifest-driven proposal naturally leads to modular and extensible architectures, and also facilitates tasks such as the exportation of contents in IMS-aware format, which can be difficult to do in other approaches. Furthermore, works on the architecture of general [34,35] and IMS-based e-learning platforms [36] usually put the emphasis on the architecture of the overall system instead of on their authoring capabilities.

Many commercial (e.g. [37,38]) and free (e.g. [39]) authoring tools support production of Web-based courses. These tools are usually conceived as standalone applications, and they are independent of particular e-learning platforms. In turn, most of the widely used e-learning platforms (e.g. [33,40,41]) include some limited authoring capabilities. In <e-Aula>, we have included a powerful Web-based authoring system in the platform itself that provides teachers with an integrated scenario in which to produce and maintain their e-learning materials. In addition, the modularity and extensibility of this authoring system enables a smooth evolution to accommodate the authoring needs arising in each specific pedagogical domain. This evolutionary nature, which is achieved by the manifest-driven approach, is also a distinctive feature of our work with respect to the aforementioned systems. In [42], an IMS-based authoring tool similar to the tool described here is presented, although no details are provided about its internal architecture and implementation, as well as about its integration (if any) with an e-learning platform.

The evolution of the authoring system is also a fundamental requirement in works such as [43,44]. Nevertheless, while those works start with an application-dependent authoring tool that evolves as new knowledge is gained in the application domain, in <e-Aula> we adopt a dual approach: we start with a general-purpose authoring system that can be specialized according to particular e-learning scenarios whenever they are encountered. Previous work on the <e-Aula> platform, including its authoring system, is described in [16,17]. Previous work on the manifest-driven approach is described in [45].

6. CONCLUSIONS AND FUTURE WORK

In this paper we have described our manifest-driven approach to architecting an IMS-based authoring system integrated in <e-Aula>, a Web-based experimental e-learning platform. This approach is the result of our efforts in the development of a system aimed at the evaluation of different e-learning standards and e-learning modular architectures. The approach leads to a highly modular and extensible integrated authoring system. This system, which is able to integrate a huge amount of disaggregated authoring and deployment facilities in the same platform, can be specialized on many different learning scenarios, which can include static, moderately interactive and highly interactive e-learning content.

DOI: 10.1002/spe

The manifest-driven approach facilitates the main tasks contemplated in the authoring system: edition and presentation, as well as importation and exportation of content packages. In addition, this approach leads to an architecture that facilitates extensibility. Adding support for a new content type is just a matter of writing the code needed to edit and play content in that format. Likewise, extending the functionality of our importation system is just a matter of writing new tasks for the agenda.

The high degree of modularity of the architecture also enhances maintainability. It is easy to find the points in the source code where any changes might be needed. Changes in most modules will have a slight impact on the rest of the system. While the complexity of the architecture is high (in terms of the number of classes and files in the resulting implementation), this is the price to pay for achieving a very high degree of modularity, and therefore better extensibility and easier maintenance.

Our Java-based implementation preserves the benefits of the architecture and adds a high degree of robustness. However, we should point out as a drawback the need for more computing power in the server when compared to lighter applications based on scripting solutions (such as PHP [46]). While the choice between a large and robust system and a lightweight application will depend on the exact needs of each learning environment, the J2EE/Struts-based solution is well suited for our purposes because it allows for a continuous evolution of the system to accommodate our evaluation and research needs.

Currently, the architecture based on the manifest-driven approach is fully implemented in the <e-Aula> system. The authoring system is fully functional and adheres strictly to the IMS CP standard. As a future work, we are planning to incorporate support for more elaborate types of contents. For this purpose, we want to integrate in the system a *manual-oriented* metaphor for the production of learning content in technical domains [47,48]. This metaphor encourages organizing the learning materials into manuals of technical subjects following a suitable set of guidelines. We have implemented this metaphor using DocBook [49], and we are planning to integrate the resulting federated tools as a whole in the <e-Aula> authoring system. In this research on content production and maintenance, we also want to pay special attention to interactivity. For this purpose, we are currently working on the incorporation of authoring support for educational-oriented graphical adventure videogames. The resulting project is called <e-Game>, and some preliminary results were published in [50,51]. We also expect to incorporate content in the domain of language processors and compiler construction. In order to let students test the examples, we want to integrate the tool for prototyping language processors that is described in [52] using appropriate players. Finally, we wish to exploit the possibility of connecting external tools to deal with this kind of interactive and dynamic contents. Another research goal is to integrate support for the authoring of activity-oriented learning strategies [13]. This effort will provide another more sophisticated alternative based on the IMS LD to the current edition and playing of simpler IMS organizations. For all these initiatives, we will further envision applying the *document-oriented* approach promoted in [21–24] for both the incremental definition of new types of resources and the incremental construction of their associated handlers as processors for domain-specific descriptive markup languages. Finally, we are planning to involve field experts in the authoring of high-quality content that fully exploits the functionalities of the *eAulaAP* application profile.

ACKNOWLEDGEMENTS

The Spanish Committee of Science and Technology (TIC2002-04067-C03-02, TIN2004-08367-C02-02 and TIN2005-08788-C04-01) has partially supported this work. We also would like to thank Javier López-Moratalla for his collaboration in earlier stages of this work.

DOI: 10.1002/spe

REFERENCES

1. Collier G. e-Learning Application Infrastructure. *White Paper*, Sun Microsystems, 2002.
2. Collis B, Strijker A. Technology and human issues in reusing learning objects. *Journal of Interactive Media in Education (Special Issue on the Educational Semantic Web)* 2004; **4**. Available at: http://www-jime.open.ac.uk/2004/4.
3. Schlusmans KHLA, Koper EJR, Giesbertz WJ. Work processes for the development of integrated e-learning courses. *Integrated eLearning*, Jochems W, van Merrienboer J, Koper EJR (eds.). Routledge Falmer: London, 2003; 126–138.
4. Instructional Management System Global Consortium. http://www.imsglobal.org [March 2006].
5. Faulkner Information Systems. *Advanced Distributed Learning—Shareable Content Object Reference Model (ADL-SCORM)*. Faulkner Information Services: Pennsauken, NJ, 2003.
6. Aviation Industry Computer Based Training Committee. http://www.aicc.org [March 2006].
7. IEEE Learning Technology Standards Committee. http://ltsc.ieee.org [March 2006].
8. IMS Content Packaging Specification v1.1.4. 2004. http://www.imsglobal.org/content/packaging [March 2006].
9. IMS Question & Test Interoperability v 2.0. 2005. http:// www.imsglobal.org/question [March 2006].
10. IMS Learning Information Package Specification v1.0.1. 2005. http://www.imsglobal.org/profiles [March 2006].
11. IMS Simple Sequencing Specification 1.0. 2003. http://www.imsglobal.org/simplesequencing [March 2006].
12. IMS Learning Design 1.0. 2003. http://www.imsglobal.org/learningdesign [March 2006].
13. Koper R, Tatersall C (eds.). *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training*. Springer: Berlin, 2005.
14. Fernández-Manjón B, Fernández-Valmayor A. Improving World Wide Web educational uses promoting hypertext and standard general markup languages. *Education and Information Technologies* 1997; **2**(3):193–206.
15. <e-Aula> project. http://eaula.sip.ucm.es/aulaVIMS [March 2006].
16. Fernández-Manjón B, Sancho P. Creating cost-effective adaptive educational hypermedia based on markup technologies and e-learning standards. *Interactive Educational Multimedia* 2002; **4**:1–11.
17. Sancho P, Manero B, Fernández-Manjón B. Learning objects definition and use in <e-Aula>: Towards a personalized learning experience. *Edutech: Computer-Aided Design Meets Computer Aided Learning*. Kluwer Academic: Dordrecht, 2004; 177–186.
18. Bray T, Paoli J, Sperberg-McQueen CM, Maler E (eds.). *Extensible Markup Language (XML) 1.0* (2nd edn). W3C Recommendation, 2000.
19. Hodgings W (chair). *IEEE Standard for Learning Object Metadata*. IEEE Standard 1484.12.1-2002, 2002.
20. ADL Technical Team. SCORM 2004 Photoshop Examples Version 1.1. http://www.adlnet.org [March 2006].
21. Sierra JL, Fernández-Manjón B, Fernández-Valmayor A, Navarro A. Document-oriented construction of content-intensive applications. *International Journal of Software Engineering and Knowledge Engineering* 2005; **15**(6):975–993.
22. Sierra JL, Fernández-Valmayor A, Fernández-Manjón B, Navarro A. ADDS: A document-oriented approach for application development. *Journal of Universal Computer Science* 2004; **10**(9):1302–1324.
23. Sierra JL, Fernández-Valmayor A, Fernández-Manjón B. A document-oriented paradigm for the construction of content-intensive applications. *The Computer Journal* 2006. DOI: 10.1093/comjnl/bx1008.
24. Sierra JL, Navarro A, Fernández-Manjón B, Fernández-Valmayor A. Incremental definition and operationalization of domain-specific markup languages in ADDS. *ACM SIGPLAN Notices* 2005; **40**(12):28–37.
25. Coombs JH, Renear AH, DeRose SJ. Markup systems and the future of scholarly text processing. *Communications of the ACM* 1987; **30**(11):933–947.
26. Abelson H, Sussman GJ. *Structure and Interpretation of Computer Programs* (2nd edn). MIT Press: Cambridge, MA, 1996.
27. Bodoff S, Armstrong E, Ball J, Carson D. *The J2EE Tutorial* (2nd edn). Addison-Wesley: Reading, MA, 2004.
28. Goodwill J, Hightower R. *Professional Jakarta Struts*. Wrox Press: Birmingham, 2003.
29. Krasner GE, Pope TS. A description of the model-view-controller user interface paradigm in the smalltalk 80 system. *Journal of Object Oriented Programming* 1988; **1**(3):26–49.
30. Chopra V, Eaves J, Jones R, Li S, Bell J. *Beginning JavaServer Pages*. Wrox Press: Birmingham, 2005.
31. World Wide Web Consortium. *Document Object Model (DOM) Technical Reports*, World Wide Web Consortium, 2004.
32. Clark J (ed.). XSL Transformations (XSLT) Version 1.0. *W3C Recommendation*, World Wide Web Consortium, 1999.
33. WebCT system. http://www.webct.com [March 2006].
34. Anido-Rifón L, Fernández-Iglesias J, Llamas-Nistal M, Caeiro-Rodríguez M, Santos-Gago J, Rodríguez-Estévez JS. A component model for standardized Web-based education. *ACM Journal of Educational Resources in Computing* 2001; **1**(2). DOI: 10.1145/384055.384056.
35. Avgeriou P, Papasalouros A, Retalis S, Skordalakis M. Towards a pattern language for learning management systems. *Educational Technology & Society* 2003; **6**(2):11–24.
36. Torres da Silva V, Pereira de Lucena CJ, Fuks H. ContentNet: A framework for the interoperability of educational content using standard IMS. *Computers & Education* 2001; **37**:273–295.
37. Lersus System. http://www.lersus.com/ [March 2006].

DOI: 10.1002/spe

38. Wilson S, Thornton J. *Authorware 6*. Thomson Delmar Learning: Florence, KY, 2001.
39. Bolton University (2005) Reload Project. http://www.reload.ac.uk/ [March 2006].
40. Dokeos. http://www.dokeos.com [March 2006].
41. Moodle. http://www.moodle.org [March 2006].
42. Zongkai Y, Gang Z, Di W, Jianhua H. A standard visual courseware authoring tool based on content packaging specification. *Proceedings of the 2004 International Conference on Information Technology: Coding and Computing (ITCC'04)*, Las Vegas, NV, 5–7 April 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004.
43. Fernández-Valmayor A, Guinea M, Jiménez M, Navarro A, Sarasa A. Virtual objects: An approach to building learning objects in archeology. *Computers and Education: Towards a Lifelong Society*, Llamas-Nistal M, Fernández Iglesias MJ, Anido-Rifon LE (eds.). Kluwer Academic: Dordrecht, 2003.
44. Sierra JL, Fernández-Valmayor A, Guinea M, Hernanz H, Navarro A. Building repositories of learning objects in specialized domains: The chasqui approach. *Proceedings of the 5th IEEE International Conference on Advanced Learning Technologies (ICALT'05)*, Kaohsiung, Taiwan, 5–8 July 2005. IEEE Computer Society Press: Los Alamitos, CA, 2005.
45. Sierra JL, Moreno Ger P, Martínez Ortiz I, López Moratalla J, Fernández-Manjón B. Building learning management systems using IMS standards: Architecture of a manifest-driven approach. *Proceedings of the 4th International Conference on Advances in Web-Based Learning (ICWL 2005)* (*Lecture Notes in Computer Science*, vol. 3583), Hong Kong, China, 31 July–3 August 2005. Springer: Berlin, 2005; 144–156.
46. Lecky-Thompson E, Eide-Goodman H, Nowicki S, Cove A. *Professional PHP5*. Wrox Press: Birmingham, 2004.
47. Martínez-Ortiz I, Moreno-Ger P, Sancho-Thomas P, Fernandez-Manjón B. Using DocBook to aid in the creation of learning content. *New Trends and Technologies in Computer-Aided Learning for Computer-Aided Design*, Rettberg A, Bobda C (eds.). Springer: New York, 2005; 11–24.
48. Moreno-Ger P, Martínez-Ortiz I, Sierra JL, Fernández-Manjón B. A descriptive markup approach to facilitate the production of e-learning contents. *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies (ICALT'06)*, Kerkrade, The Netherlands, 5–7 July 2006. IEEE Computer Society Press: Los Alamitos, CA, 2006.
49. Walsh N, Muellner L. *Docbook: The Definitive Guide*. O'Reilly: Sebastopol, CA, 1999.
50. Martínez-Ortíz I, Moreno-Ger P, Sierra JL, Fernández-Manjón B. Production and maintenance of content-intensive videogames: A document-oriented approach. *Proceedings of the 3rd International Conference on Information Technology: New Generations (ITNG'06)*, Las Vegas, NV, 10–12 April 2006. IEEE Computer Society Press: Los Alamitos, CA, 2006.
51. Moreno-Ger P, Martínez-Ortiz I, Fernández-Manjón B. The <e-Game> project: Facilitating the development of educational adventure games. *Cognition and Exploratory Learning in Digital Age (CELDA'05)*, Porto, Portugal, 14–16 December 2005. IADIS, 2005.
52. Sierra JL, Fernández-Valmayor A. A prolog framework for the rapid prototyping of language processors with attribute grammars. *Proceedings of the 6th Workshop on Language Descriptions, Tools and Applications (LDTA 2006)*, Vienna, Austria, 1 April 2006. *Electronic Notes in Theoretical Computer Science* (in press).

DOI: 10.1002/spe