Component-based implementation of tool e-package

Yassine Aarab, Noura Aknin, Abdelhamid Benkaddour

Abstract— We are trying in this article to present the process of development of a prototype application entitled e-package: a tool of description and packaging of components. We will present the services it offers and its principle operations. As well, we'll present its architecture that it integrates a system of indexing and search of XML documents fully configurable. We will present the technological choices of its implementation.

Index Terms— Component, Components interfaces, Educational Software, Metadata, Mechanisms of composition of components, Software Components, XML , Warehouse

1 Introduction

The complexity of the educational requirements imposed on the development of software a series of demands that allow them efficiently, with quality and with a reasonable cost-benefit ratio for the development of the industry and a fair price for the education sector. However, the reality of educational software development is described as a critical [1] for various reasons:

- Excessive cost of development, in terms of both ressources and time that is difficult to estimate in advance.
- Poor quality in the development process as well as of the final product.
- Lack of capacity to adapt to changing requirements, quickly and efficiently.

To resolve this problem in part the need has arisen to modify the *software development process* [2] toward greater compositional reuse; adopting a model of software development based on components.

In this article, we present the case of educational software called e-package that adapt to new educational requirements. Given the problem, attention is drawn to the transformation process using a component-based architecture of software; indicating their main advantages and limitations detected in the process.

2 Context

Our goal is to implement a warehouse of software components. For that purpose, we have built a diagram of metadata to describe, a boss to create profiles for applications, and a generator of interface to build the form of filling of the metadata [3]. All these elements are gathered in a package ECR (Educational Component Repository). The metadata are used to find the components, furthermore they must be attached. Thus, we propose a format of package to exchange and share software components: E-package. We present, subsequently, the approach to follow to build this package and we will provide a prototype tool for metadata and generation of package.

- *Yassine AARAB: currently pursuing phd's degree program in Abdelmalek Essaâdi University, Morocco, PH-00212661222241. E-mail: yacinorock@gmail.com*
- *Noura AKNIN is professor at Abdelmalek Essaâdi University, Morocco,. E-mail: aknin@ieee.com*
- Abdelhamid Benkaddour: *is professor at Abdelmalek Essaâdi University, Morocco. E-mail : ham.bekaddour@gmail.com*

The package ECR will contain the software components developed by the Community e-learning. Each package of component is formed of a body and a description. The body is composed of executable binary code, the design model, source code, games testing, documentation, etc. The description contains the metadata defined in the scheme of metadata. Overall, the warehouse is composed of two parts: a catalog and a set of software components. The catalog includes the metadata of the components removed and information on their location. The components are archived in a file system.

To be operational, the warehouse must provide, to its users, if possible all the following features:

- Identification and description of a software component. This information comes from the metadata and is recorded in the catalog.
- Publication of software entities allowing developers to insert in the warehouse a new software component.
- All course of the catalog allowing users to consult the description of the software component desposited in the warehouse.
- Recovery of a software component that allows users to obtain in their work environment a copy of the body of software component sought.
- Classification and search. In the case of a large warehouse, the course of sequential catalog is not enough. It must have the features of text search, by keyword, and by attributes.

3 Theoretical Framework

3.1 Model of the descriptor of package

To be treated properly, the contents of the manifest.xml file must be valid in relation to the model of the descriptor of given package. The following figure illustrates the structure (figure 1):

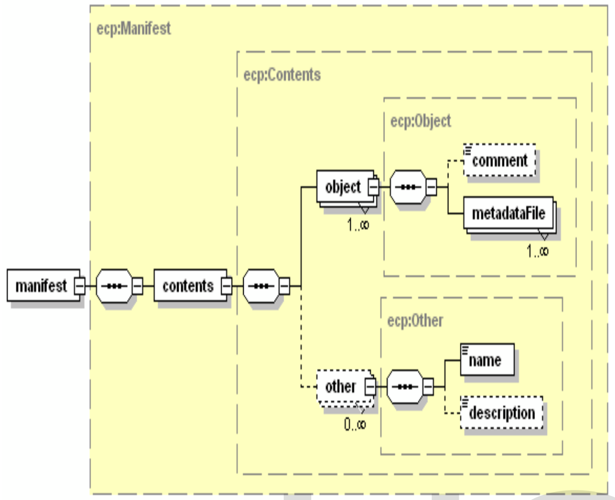

Fig. 1. Diagram of the model of the descriptor of the ECR package

The object element, in this model is a general practitioner. It is not typed. It is used to declare any object of the package described by metadata [4]. For the categorization, we must assign to its attribute one of the following values:

1. "SoftwareComponent" to indicate the object in quesion is a software component;
2. "Software" if the object is an exploitable software;
3. "EducationalRessource" if the object is a pedagogical resource;
4. "LearningObject" if the object is an educational object in the common sense of the e-learning community;

In addition to this attribute, the object element has two optional attributes (i.e. id and name) to identify it without going through the metadata.

To reference a metadata file associated with the object, it must instantiate the Child element metadataFile. The latter includes, in addition to the optional attribute id, three mandatory attributes: type, pathRoot and URL.

The attribute type should, indicate the name of the schema of metadata used to describe the object. The allowable values for the moment are: LOM [5] and DublinCore [6]. This list may evolve to integrate new schema names. The URL of the schema is located in the root element of the XML document containing the metadata.

The attribute pathRoot should, indicate if the URL of the metadata file is absolute or relative. The possible values are:

5. "This" indicating that the URL is relative to the manifest.xml file;
6. "package" indicating that the URL is relative to the root folder of the package;
7. "absolute" indicating that the URL is an absolute reference;

Finally, the URL attribute should, mention the address of the metadata file.

As we can see, with the cardinalities of the metadataFile element, it is possible to describe the same object with several metadata schemes. Each instance of metadataFile corresponds to a metadata file. The supplier of the component must ensure the consistency of the values assigned to attributes with the actual content of each file.

The content has also another optional element allowing to referencing other resources of the package that are not accompanied by metadata. The two child elements name and description allow the indication of what resource it is. Other resource has the same attributes as the object, with the exception of several values are not associated to the type attribute and the attribute URL locates this time the resource.

3.2 Build a package: Methodology

There are two ways to build a package: either manually or with a dedicated tool. In this section, we show the limits of the manual approach and the need to have a specialized tool simplifying the task of construction of the ECR.

3.2.1The manual approach

Given the characteristics of the package ECR, it is not very difficult to build it manually. A simple XML editor and a compression utility are sufficient to obtain it. This approach can be used when the task is occasional and there are only one or two software components in the package, which is rarely the case in a large project. The major difficulty lies not in the creation of the package but rather in the filling of metadata. By experience, it quickly becomes tricky to manipulate and navigate in an XML document as soon as its size becomes impressive. From a few hundreds of lines, without XML editor sophisticated, the document becomes unmanageable. However, the schema has a lot of metadata. Its XML instances are difficult to handle and can easily become aversive to inform even for the more experienced in XML [7].

It is obvious that without the tool of capture of the metadata and packaging, all the experiments that we can plan are doomed to failure. To succeed in this phase, it must be put at the disposal of users responsible to describe the software components a tool that facilitates their task.

3.2.2 The instrumental approach

This approach is to operate an application to populate the metadata, build and generate the package ECR. The objective of this type of tool is to simplify to the maximum the

phase of description of the components and construction of the package, decreasing the burden of work for the users.
We developed a tool offering the following features:

- Filling of metadata through forms.
- Construction of the package by adding file or URL link.
- Provision of a tree view of the package with the features CRUD (Create, Read, Update, and Delete).
- Generation and integration of the manifest in the package.
- Backup and reading of the package in the appropriate format.
- Transfer by Web service of the package to a server.

At the level of forms of filling metadata, we want to manage the descriptors defined in the application profile. The objective is twofold: firstly, we want to reduce the burden of description of the components, exploiting the descriptors provided in the model part of the Application Profile. To achieve that, we assign the values by default or the Association lists of values to some metadata. Secondly, we want to ensure the control of entered information and deliver presentations tailored to the specialty of the users. The nature of the Metadata requires, in effect, the participation of users of different specialties: Designer, assembler, librarian, etc. For each specialty, it must show the appropriate metadata with adequate instructions. The objective is to guide users in their task of description without polluting their interface with unnecessary information and overloading their cognitive effort.

Thanks to this type of tool we hope to offer a quick and more user-friendly construction of ECR packages. We are convinced of the need to invest in such projects because we cannot request from the Community to participate in the mutualisation of software components without providing them with the necessary means to achieve this objective.

4 Architecture of e-package

In this section, we present the architecture of the tool e-package that we propose to the community to build its ECR packages. e-package is an application capable, from the parameter data, to provide the user with a graphical user interface to populate the metadata and build its package.

E-package is an assembly of three large components. The first is responsible for building package, the second for providing the metadata entered by the user in the form of XML docment and the third for shipping by Web Service a file to a server.

When we start the application, a parameter file is automatically loaded. The propreties of the structure of the package are transmitted to the component packageBuilder. The URL of the metadata application profil is transmitted to the component IG. The propreties of the Web Connection Service are transmitted to the WebService Component. Each component is then responsible to initialize its environment. The graphical interface is then displayed.

4.1 The packaging

In the inistializing the Component *packageBuilder*, the structure of the package is saved in the form of tasks to perform during the backup. A graphical presentation of this structure is presented in the form of a tree. The user has, then, of the crud features to create directories, add documents or files savedbon its file system. It also has the possibility to add URL of documents or files available online. All actions are saved as tasks to achieve during the backup.

4.1.1 The filling of metadata

This activity is ensured by the component IG [1]. It is in charge of providing from the information defined in the Application Profile, the necessary forms to seizure of the metadata. Through these forms, the user enters the metadata. The information you entered are saved in memory pending their registration in files.

4.1.2 Generation of the package and transmission to the server

When the user requests to save the package, a temporary folder is created in the file system. Directories requested during the creation of the package are created in the temporary folder and files are copied there.

Then, the metadata are stored in XML files under the folder metadata. The manifest file is subsequently generated in this same folder. This file are referenced the metadata files. The objects are referenced in the package and the URLs of the files and documents are available online.

The package is finally obtained by Zip compression of this temporary folder. The user can save locally in assigning a name or send it to the server.

If the package was sent to the server, the WebService component prepares a SOAP message [8]. Then the message will be forward as attached file in the package and establishes a web connection service with the server via the RPC protocol. The server is supported by the following to receive the message and store the package in view of its treatment.

5 Conceptual Framework

Below are the main theoretical aspects of the software components, which were considered in the development of a version of e-package based in software components.

5.1 Definition

Although there are a variety of definitions with regard to what is a software component, there is some consensus to define it as a set of objects that meet a specific function, specify interfaces and operate independently.the software components can be composed of other components and be used to create software systems of greater complexity [9, 10]. Their main characteristics are summarized below [11]:

- **Composiion:** It is the capacity that allows you to integrate components of greater granularity. To do this, the interactions must occur at the level of component interfaces.

- **Encapsulation:** the component must be able to hide details of its implementationand function as a true black box.
- **Interoperability**: The component is interoperable when it can work independently and can interact with other components to deploy a functionality of greater complexity
- **Multiplatform:** To promote the reuse, it is desirable that the component can work independently of the hardwaeand the operating system.
- **Auto-content:** The component is self contained when depends at least possible other to fulfil its purpose. To do this, it must present a low coupling and a high cohesion.

5.2 Component Interfaces

An essential aspect in the consruction of software component are the component interfaces (API). These are mechanism that allows its interoperability [11]. To do this, the component interfaces present an “interface provides”message that declares a set of services that the component implements an "interface requires" message that specifies the services required for the component to operate, declaring a set of events that the component can issue.

Events enable to communicate a response to an external stimulus or a change in the internal state of the component. At the interface of the component we find a symbol which specifies the event.

5.3 Composition of components

According to Sametinger the composition of components is in the process of building applications using the interconnection of software components through their interfaces [12]. Seen in a practical way, this process can be understood as a client-server relationship. The client component requests a service that is offered within the operations defined in the component interface server. Then the server component runs the required operation and returns the results to the client synchronously. From the point of view of the interaction between the interfaces of the components, three types of composition of components can be distinguished.

- **Sequential Composition:** occurs when there is incompatibility between the component interfaces to interconnect. This requires an adapter component to reconcile these interfaces.
- **Hierarchical composition**: occurs when a component performs a request directly to the services provided by another component, not existing incompatibilities between them.
- **Additive composition:** occurs when the interfaces of two or more components come together to create a new component of greater granularity.

6 IMPLEMENT OF E-PACKAGE

6.1 Development based in components of software

OMT++ is one of the methodologies of analysis and design that is Object-oriented, more mature, efficient and widely used at present for the development of software projects. It aims at getting the specification, design, and implementation correct at the outset [13]. Iteration is reduced and maintainability of documentation is improved by delaying the decisions to the points where all essential informations are available. This paper also presents a notation for specifying user interfaces.

This process presents modifications with respect to the development of traditional software mainly for the stages of analysis, design and development.

In the analysis phase, it is entered the activities of search, selection and adaptation of existing components for reuse in the component-oriented development.

In the design phase component-oriented, introduces a new phase called Design of interfaces in which clarifies the existence of interfaces, identifies the interfaces that can be reused and discovers new interfaces and operations.

The development phase is divided into the processes of: development of components and integration of components. The process of development is performed in the case of existing resources that are not suitable to be reused and requirements that have not been able to be satisfied by the repository.

The process of integration is done on the basis of a set of components always trying to maximize its reuse and thus reduce their number that need to be developed from the start. To achieve this, it should have repositories that are reusable, reliable and to act according to their specifications. This integration is done considering the interfaces of components and the composition methods already described.

6.1 Development based in components of software

The principles components and their interfaces are deducted from class diagram at Fig.2:

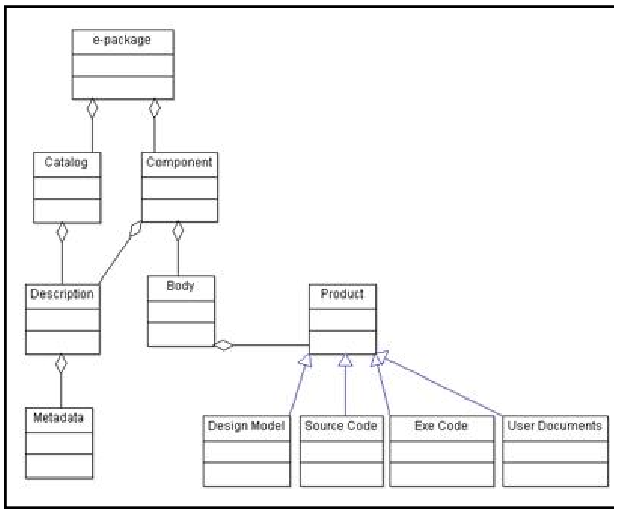

Fig. 2. Class diagram e-package

Once the class diagram was implemented, proceeded to establish the set of components that would serve as the building blocks of the e-package as shown at Fig.3

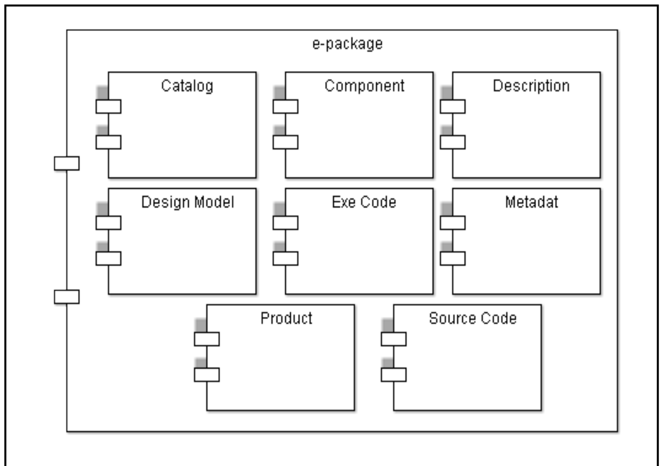

Fig. 3.Initial component diagram of e-package

6.3 Design of e-package interfaces

On the basis of the architectural design, all components implemented and to implement must be accessed through its interface of components and they will be forced to specify the following services:

- **Start (conf:XML): void:** This service allows you to configure the components contained in e-package using a configuration in XML format that is broken down as you go down the granularity of the component.
- **setSession (session: XML): void, getSession (): XML: reset (): void:** These services allow the handling of the meetings of the components.
- **getId (): INT, getDescription (): String:** These services allow you to identify the components in regard to your id and description.
- **READY_COMPONENT:** event that notifies the success of the burden of the sub-components of a component.
- **FAILURE_COMPONENT**: event that notifies the fault of the burden of the sub-components of a component.

6.4 Detailed design of e-package

The component e-package is the essential piece of software for the implementation of the new version of e-package because it is responsible for loading, configuration and deployment of others components. In addition, this component interoperates with the ECR that contains it to support the functionality to open, save, create, export and print a session are described below.
The internal modeling of the e-package components is presented at Fig.4.

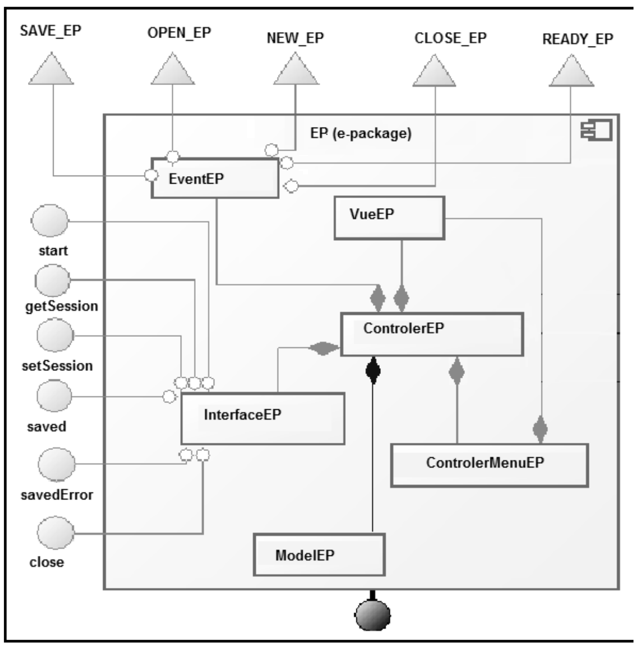

Fig. 4.JavaBeans component e-package

The mechanism of composition of components is defined by the service start that is implemented by the class "ControlerEP". This service receives as a parameter settings in XML format that details aspects of loading and configuring the components contained in e-package.

```
        Public function Start (conf: XML=""):void{
                this.conf = conf;
                this.downloadComp();
}
```

The service start executes the method downloadComp that is responsible for loading each of the components defined in the XML configuration. To do so is obtained from the XML configuration the path of the component and then accesses the modelEP class to perform the load of the component from the repository of components.

```
        Private function downloadComp () : void{
                Var path = conf.adress;
                ModelEP.downloadComp (Path);
                }
```

The downloadComp method invokes the method *loader* who is responsible for loading the component as a MovieClip object.

```
        Private function downloadComp(path:String){
                loaderContext =new LoaderContext();
                ComponentEpLoader = new Loader();
                ComponentEpLoader.load (new URLRequest
                (path, loaderContext);
                }
```

The *load function* performs the burden of packaged component within a *MovieClip* in asynchronous mode. Once the virtual machine JVM performs the load of the component, this emits an event that is heard by the class modelEP by defining the listener initHandler.

```
Private function initHandler (event:event) :void{
        comp = event.target.content;
        controlerEP.setStrategy (comp);
}
```

Once you have loaded the component is made a call to the method setStrategy which is the placing of two *listeners* to be able to handle the load of the components. Then invoked the service *start* the component charged delivered as a parameter the XML settings related to the strategy reader that it implements.

```
Privatefunctoin setStrategy (comp:MovieClip) : void
        {
        This.compActual = comp;
        comp.addEventListner;
        ("Ready_Component", exitdownload);
        comp.addEventListner;
        ("Failure_Component", downloadErr);
        Comp.start (configuration.strategy [with t]);
        }
```

If the components of the strategy reader were loaded successfully activates the listener exitodownload, which is responsible for storing the component in an arrangement of components and then continue with the normal loading of the following components defined in the XML configuration.

```
Public function exitdownload (e:event) : void{
        arrangeComponent.push (compActual);
        keepDownloadingComponent (true);
        }
```

In the case of there an error in the loading of the component of type reader, strategy activates the listener downloadErr which is responsible to continue with the load of the following components.

```
Public function downloader (e:Event) : void{
        keepDownloadingComponent (false);
        }
```

7 Conclusion

From the results obtained it was possible to conclude that the component-oriented development presents several advantages for the development of educational software, allowing the reuse of software assets, either existing or in the development of future applications.

References

[1] Butcher, N. (2015). A basic guide to open educational resources (OER). Commonwealth of Learning, Vancouver and UNESCO.

[2] Barrett, S., Higgins, C., Twomey, C., & Evans, M. (2006). U.S. Patent No. 7,000,219. Washington, DC: U.S. Patent and Trademark Office.

[3] Aarab Y, Aknin N, & Benkaddour A. (2016)." Generation System of Metadata Software Components: A Proposed Architecture". Proc. the Mediterranean Conference on Information & Communication Technologies 2015, Vol. 381 of the series Lecture Notes in Electrical Engineering,pp313-322.Available. http://link.springer.com/chapter/10.1007%2F978-3-319-30298-0_33

[4] Foulonneau, M., & Riley, J. (2014)." Metadata for digital resources: implementation, systems design and interoperability". Elsevier.

[5] Neven, F., & Duval, E. (2002, December). "Reusable learning objects: a survey of LOM-based repositories". Proc. The tenth ACM international conference on Multimedia pp. 291-294. ACM

[6] Dublin Core Metadata Initiative. (2014). Dublin core metadata initiative.

[7] Boberic, D., & Surla, D. (2009)." XML editor for search and retrieval of bibliographic records" in the Z39. 50 standard. The Electronic Library, 27(3), pp. 474-495.

[8] Roy, J., & Ramanujan, A. (2001)." Understanding web services". IT professional, 3(6), pp. 69-73.

[9] Szyperski C. (1998). Component Software Beyond Object–Oriented Programming. Edinburgh Gate: Addison–Wesley.

[10] Broy M., Deimel A., Henn J., Koskimies K., Plasil F., Pomberger G., Pree W., Stal M. & Szyperski C. (1998). What characterizes a (software) component? Software, Concept and Tools. 19(1) pp. 49-56.

[11] Crnkovic I. & Larsonn M. (2002). Building reliable component-based software systems. Boston: Artech House.

[12] Sametinger J. (1997). "Software Engineering with reusable components". Berlin: Springer-Verlag.

[13] Forsell, M., Halttunen, V., & Ahonen, J. (2000, June). "Use and identification of components in component-based software development methods". Int.Conf.Software Reuse. pp. 284-301. Springer Berlin Heidelberg.