Authoring game-based adaptive units of learning with IMS Learning Design and <e-Adventure>

Daniel Burgos*

Educational Technology Expertise Centre (OTEC)
Open University of the Netherlands
Valkenburgerweg, 177
6419AT Heerlen, the Netherlands
E-mail: daniel.burgos@ou.nl
*Corresponding author

Pablo Moreno-Ger, José Luis Sierra and Baltasar Fernández-Manjón

Department of Software Engineering and Artificial Intelligence
Complutense University of Madrid
Profesor José Garcia Santesmases s/n
28040 Madrid, Spain
E-mail: pablom@fdi.ucm.es
E-mail: jlsierra@fdi.ucm.es
E-mail: balta@fdi.ucm.es

Rob Koper

Educational Technology Expertise Centre (OTEC)
Open University of the Netherlands
Valkenburgerweg, 177
6419AT Heerlen, the Netherlands
E-mail: rob.koper@ou.nl

Abstract: Electronic games and simulations (eGames) are a valuable support for adaptive learning. This adaptation can be based on different inputs, such as the user's performance, behaviour or cognitive load. Both adaptation and eGames can be modelled with IMS Learning Design (IMS LD) or integrated from an external resource. In this article we show the relation between IMS LD and the <e-Adventure> Project when it comes to authoring adaptive Units of Learning (UoLs) integrated with eGames. We first describe the challenges of this objective and the several different solutions on authoring and integration. We also describe the content-centred authoring approach in <e-Adventure>, and the need for a communication service with IMS LD that makes a bidirectional influence on the user's adaptive learning experience. At the end, we describe a practical example that illustrates how an adaptive IMS LD UoL with an integrated <e-Adventure> eGame is developed.

Keywords: authoring; adaptive learning; IMS Learning Design; IMS LD; <e-Adventure>; game-based learning; learning technology.

Reference to this paper should be made as follows: Burgos, D., Moreno-Ger, P., Sierra, J.L., Fernández-Manjón, B. and Koper, R. (2007) 'Authoring game-based adaptive units of learning with IMS Learning Design and <e-Adventure>', *Int. J. Learning Technology*, Vol. 3, No. 3, pp.252–268.

Biographical notes: Dr. Daniel Burgos works as an Assistant Professor at the Open University of the Netherlands, after having worked 14 years as a Teacher, Multimedia and Game Developer, and Academic Manager in Europe and South America, as well as with his own company. He is mainly focused on adaptive e-learning, IMS Learning Design, eGames and learning networks, and he is involved in the research projects ProLearn and TenCompetence. More information at www.danielburgos.eu.

Pablo Moreno-Ger, MSc, is a member of <e-UCM>, the e-learning research group at the Complutense University of Madrid (UCM). His research interests include e-learning technologies and the integration of video games and simulations in learning environments. He is currently working on his PhD thesis in the Department of Software Engineering and Artificial Intelligence and leading the development of the <e-Adventure> educational game engine. More information at www.e-ucm.es/people/pablo.

Dr. José Luis Sierra is an Assistant Professor in the Department of Software Engineering and Artificial Intelligence (DISIA) at the Complutense University of Madrid (UCM), Spain. He is a member of <e-UCM>, the e-learning research group at the UCM. His research interests include e-learning technologies, domain-specific languages and mark-up languages. He received his PhD in Computer Science from the UCM. More information at www.e-ucm.es/people/jlsierra.

Dr. Baltasar Fernández-Manjón is an Associate Professor in the DISIA at the UCM, where he co-leads the <e-UCM> group. He is also the Vice Dean of Research and Foreign Relations at the Computer Science School of this university. His main research interests are e-learning technologies, educational uses of mark-up technologies, application of educational standards and user modelling. He received his PhD in Physics from the UCM. More information at www.e-ucm.es/people/balta.

Professor Dr. Rob Koper is a Full Professor of Educational Technology and the Head of the Development Programme at OTEC, Open University of the Netherlands. His research is focused on personalised instructional, web-based learning environments and self-organised distributed learning networks for lifelong learning, including the use of interoperability specifications and standards. More information at www.learningnetworks.org.

1 Introduction

IMS Learning Design (IMS LD) is focused on providing a flexible specification to model several pedagogies. Two of the main objectives of this specification are to model adaptive learning and to provide interoperability. IMS LD allows the modelling of highly adaptive Units of Learning (UoLs) in order to provide personalised learning experiences, but UoL authoring is a complex task, mainly due to the lack of high-level IMS LD

authoring tools. This problem becomes more complex when we try to integrate a UoL with external resources developed with different technologies and to establish a communication flow between both parts.

On the other hand, electronic games and simulations (eGames) are a useful resource when dealing with adaptation and e-learning. In addition to the motivational enhancements to the learning process, eGames can provide adaptive content and fully personalised itineraries. With this potential in mind, and with the objective of simplifying the authoring process for educational games, the <e-Adventure> Project provides a rich notation and an educational game engine targeted at a very specific game genre: *Point-and-Click* graphical adventure games.

Drawing on these ideas, in this article we propose an authoring approach for adaptive IMS LD UoLs with embedded eGames. Indeed, the main obstacle to the integration of external software applications and modules developed outside IMS LD is the increase in the complexity of the authoring process. We first describe IMS LD and the challenges it presents from an authoring perspective. Then we introduce <e-Adventure> and its authoring approach, which results from the marriage between descriptive mark-up languages and domain-specific languages. Next we adopt a similar authoring approach for the integration of <e-Adventure> eGames and IMS LD. All these aspects are illustrated with an example regarding an adaptive IMS LD UoL with an eGame embedded. Finally we provide some conclusions and outline some lines of future work.

2 An adaptive learning environment with IMS Learning Design and eGames

2.1 Adaptive learning and the IMS Learning Design specification

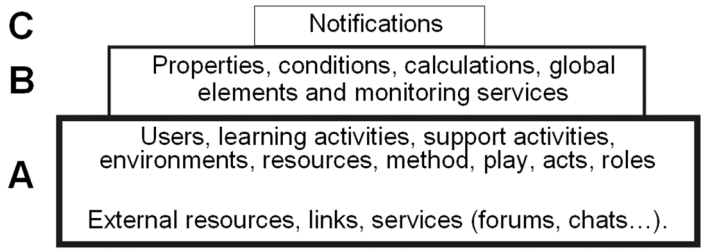
IMS LD (2003) is a specification to represent and encode learning structures and methods for learners and teachers. Furthermore, IMS LD is focused on the design of pedagogical methods able to manage learning activities linked to learning objects within a learning flow (Koper and Tattersall, 2005). This learning flow consists of plays, acts, activities, activity structures and environments and it is flexible enough to provide several personalised itineraries depending on the role assigned or on a set of rules.

The usual life cycle starts with a lesson plan modelled according to the IMS LD specification, defining roles, learning activities, services and several other elements, inside an XML document (W3C, 2003) called Manifest (Tattersall *et al.*, 2003). An information package written in IMS Content Packaging (IMSCP, 2001) is used as a container for the resources and links them with the IMS LD structure. Later, the Manifest is packaged with the nested resources in a compressed ZIP file, meaning a UoL. Several examples are available in OUNL (2002) and LN4LD (2005).

Therefore, the UoL is distributed as a compressed file with (a) an XML Manifest, describing methods, plays, acts, roles, activities, environments, properties, conditions and/or notifications of the LD specification and also pointing to the related resources; and (b) a set of files or resources mentioned in the XML Manifest. Once the UoL is validated, published and run in a player, the player will coordinate the teachers, the students and the activities during the learning process (Koper and Tattersall, 2005).

IMS LD consists of three levels (Figure 1): Level A is the main part of the specification as it provides the base line for building any UoL with the elements Method, Plays, Acts, Roles, Role-Plays, Learning activities, Support activities and Environments; Level B adds some features to create more complex lesson plans using Properties, Conditions, Calculations, Monitoring services and Global elements; and Level C adds Notifications. Each layer is built upon the previous one (Koper and Burgos, 2005).

Figure 1 IMS Learning Design and the three levels



In addition to the basic structure of Level A, the elements in Level B are actually the key for more expressive UoLs (for instance, based on adaptation or collaboration), as they combine several features that encourage and make the content and the learning flow more flexible (Koper and Burgos, 2005; Specht and Burgos, 2006). Furthermore, the combination of these elements allows for the modelling of several classical adaptive methods (*e.g.*, reuse of pedagogical patterns, adaptability, navigational guidance, collaborative learning, contextualised and mobile distributed learning, adaptation to stereotypes), making use of different structural elements of IMS LD, like Environment, Content, User groups and Learning flow.

Every single step between the creation and the use of a UoL needs an IMS LD–compliant tool. The UoLs can be created with general-purpose editors, such as XML Spy (Altova, 2006), or with specific IMS LD editors, such as CopperAuthor (Van der Vegt, 2005), Cosmos (Miao, 2005) or Reload LD Editor (Bolton, 2004); and they can be run with several tools and engines, such as CopperCore (Vogten and Martens, 2005) or SLeD (OUUK, 2005). However, current tools do not allow for easy editing and a significant effort is still needed to create adaptive IMS LD UoLs comprising level B and C constructs (Burgos *et al.*, 2006a). They make the creation of adaptive UoLs technically possible, but too difficult for a nontechnical user. A higher-level layer with a more visual metaphor is still missing, although some initiatives are being taken. For instance, the TENCompetence Project (TENCompetence, 2005) and the Complutense University (UCM, 2006) are developing a visual LD Editor each. The conception of these tools will ultimately rely on the domain concepts behind the specification, which are incarnated in the domain-specific XML binding.

2.2 Integration of eGames with IMS LD

Electronic games and simulations (eGames) engage people (Prensky, 2001; Garris *et al.*, 2002; Squire, 2002). EGames improve motivation and involvement, which results in a deeper learning experience (Malone and Lepper, 1987; Cordova and Lepper, 1996). They

provide the player with a long list of benefits: fun, interactivity, problem solving, user involvement, motivation and creativity, to mention a few (Prensky, 2001). They also awaken personal feelings and emotions in the players, such as wonder, power or aggression, and provide support for the development of personal competences like focused goals, rules, tasks, affiliation, choice, or the absence of adverse consequences in the case of a wrong choice (Squire, 2002).

EGames also provide input, output and feedback in real time (Rieber, 1996; Laurillard, 1998), which are used in adaptive learning, *e.g.*, choosing the next action to take or the contextualised help provided. In order to achieve the educational objectives, we can use various interactive learning techniques – *i.e.*, learning from mistakes, goal-oriented learning, role playing and constructivist learning (Prensky, 2001; Gee, 2003) – within and/or around the game itself. The main goal is to make the game a fully integrated activity within the whole learning process, instead of remaining an isolated stand-alone resource (Provenzo, 1991; Gee, 2005). In doing so, generic games, as well as specific educational games, can be used as an interlaced element throughout the learning experience, thus increasing the educational threshold (Burgos *et al.*, 2006c).

However, when eGames (or other external systems) are introduced in an e-learning environment, their use is often isolated from e-learning systems and other information packages (*e.g.*, IMS LD, IMS CP, SCORM) (Eskelinen, 2001; Squire, 2002; Jenkins and Squire, 2003). If we model a UoL containing several activities and one of them (for instance, Activity-Game) is a game, the game will be executed separately from the main flow. The students will play the game, but no connection is established with the previous activity or the following. Therefore, the game is incorporated to a learning flow but no further communication is established with it. In this sense, the eGame is another learning object but with a lack of interaction with the rest of the setting, and is unable to influence the learning flow. In terms of communication, there is no difference between this game and other more static resources, such as a document, a video or a link to a web page.

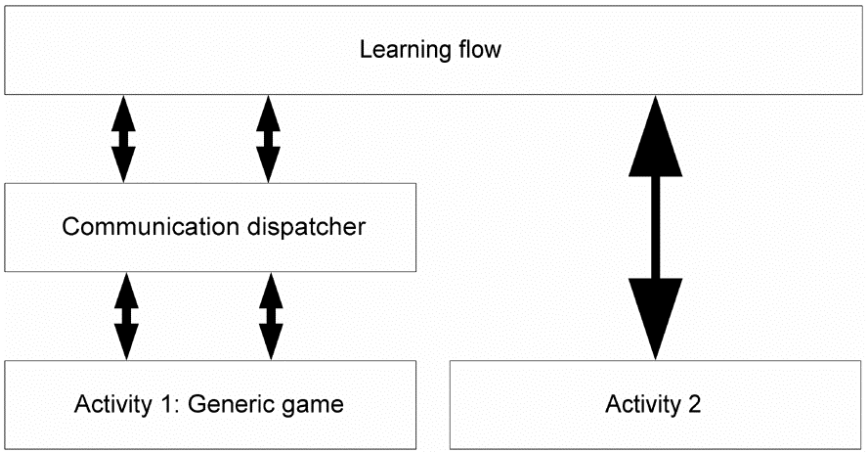
On the other hand, closer integration allows for pedagogical improvements as well as a better contextualised learning path (Richards, 2005; Burgos *et al.*, 2006c). In this approach, the activity previous to the Activity-Game can provide some input to the game. For instance, the learner answers a quiz and the final score is sent directly to the game. Then, the game could start with an adaptive setting based on this input. Thus, if the score is less than a specific threshold, the starting level is for beginners; if the score is higher, the starting level is for advanced players. During the game, a list of values of properties is sent to the learning flow to provide a detailed report after the game and/or to influence the next action to take (for instance, choosing one learning path out of several possibilities).

Therefore, the game is a fully operational part of the learning flow, able to send and receive information to and from the UoL, via a communication dispatcher, and each layer can interact with the other, hence personalising the learning experience, influencing the run-time and modifying features in both parts during the execution of the run (Figure 2).

However, this task is not necessarily easy or cost-effective. If an author wishes to include an eGame in a UoL, enabling such a communication is a major issue, as it would require modifying the source code of the executable games to support this communication. Additionally, even if the game is capable of establishing such a communication, eGames and the UoL may be developed independently, which means there has to be a connection between the variables being accounted for in the UoL and in the eGame. Thus, the rest of this work deals with how the development process for

eGames can be facilitated for content authors and how it is possible to configure (without programming knowledge) a generic communication dispatcher that translates values so that the eGame and the UoL can understand each other.

Figure 2 General architecture of communication between the learning flow and the game compared to static content. The communication dispatcher is responsible for handling the communication between the game and the learning flow



3 The <e-Adventure> Project

The main goal of the <e-Adventure> Project is to apply a documental approach (Sierra *et al.*, 2004; 2005a–b) to the authoring of educational graphical adventure video games (often also referred to as *point-and-click* adventure games). This documental approach promotes an authoring strategy that results in the marriage between descriptive mark-up languages (Coombs *et al.*, 1987), such as are used in the publishing domain, and domain-specific languages, as proposed by the software engineering community (Mernik *et al.*, 2005). Indeed, applications are described using documents. These documents are marked up with suitable domain-specific mark-up languages, and final applications are produced using suitable application generators. This approach has proven very useful in enabling the authoring by domain-experts of many different content-intensive applications in many different domains.

Having identified the high development cost and the requirement of programming skills as two of the main shortcomings in authoring educational games, the key objective of <e-Adventure> is to allow an author without a strong technical background to produce and maintain an entire game as a document using an easy-to-understand language, which is then fed to an interpreter (the <e-Adventure> engine) that produces a fully functional game.

The language is an XML application – *i.e.*, an XML-based mark-up language defined with an XML document grammar: a DTD or a Schema. The author uses this language to describe the environment, characters, objects and situations that form the game, just as another author could use a more conventional mark-up language (*e.g.*, LaTex or Docbook) in preparing a manuscript. The objective is to allow an author

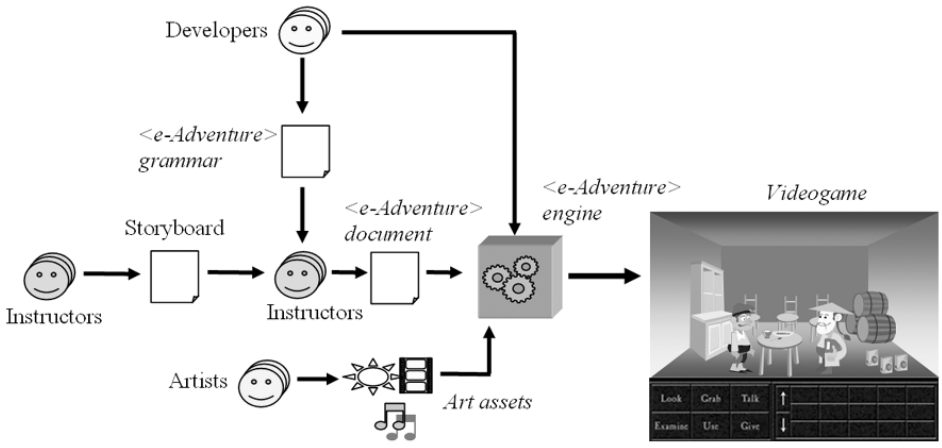
to build an executable game without needing previous background in programming. All he/she would need is basic knowledge on how to use a computer, a text editor, a few notions of XML and familiarity with <e-Adventure> syntax. In this section we will give a high-level description of the philosophy behind the project. Further details on <e-Adventure> (formerly known as <eGame>) are described in Martinez-Ortiz *et al.* (2006) and Moreno-Ger *et al.* (2006; 2007).

3.1 The authoring process in <e-Adventure>

Writing a document in the <e-Adventure> language should not feel like programming, but more like writing a story. This means that the author does not specify how the characters move or how the lighting works, but what the actual content of the game is (scenarios, items, conversations, *etc*.). Regardless of whether one uses a plain text editor or a more user-friendly editor with a lot of eye-candy, in our opinion this is the true advantage of <e-Adventure> for authoring: developing an eGame is equivalent to writing a document.

As depicted in Figure 3, the authoring process in the context of <e-Adventure> begins with the creation by the Instructors of a storyboard in natural language. There are a number of guidelines for the creation of the storyboard that facilitate the rest of the process as described in Moreno-Ger *et al.* (2007). Next, the Instructors, following the <e-Adventure> document grammar provided by the Developers (experts in computer science), add the mark-up directly onto the original storyboard. The result is an XML document, called the <e-Adventure> document, which makes the structure of the original storyboard explicit. This document will also include references to several art assets (such as graphics, animations or music) provided by a third kind of experts (the Artists), and which will be used in the production of the final video game. The <e-Adventure> document, together with the required art assets, is directly fed to the <e-Adventure> engine (provided and maintained by the Developers) for their execution. Notice that the support of Artists and Developers brings to the approach the flexibility of a full-featured custom solution to the production of video games (indeed, the authoring environment can be specialised for each production scenario) while also preserving the authoring advantages of a domain-specific solution.

Figure 3 Authoring in <e-Adventure>



3.2 The <e-Adventure> language

The <e-Adventure> language closely mirrors the typical structure that occurs in a storyboard for a point-and-click adventure game (see Figure 6 in Section 5.2 for an example of an <e-Adventure> document fragment). Following the traits of the genre, the basic unit of construction is the 'scene'. An <e-Adventure> storyboard (and thus the marked document) starts by describing all the scenes that make up the game, including the associated resources, their connections to other scenes and the description of the characters and objects that populate the scene.

The definition of items and characters is also very straightforward, focusing on their descriptions and the possible interactions that can be performed on them. An item can be roughly examined (a brief description), examined in detail (a more detailed description), combined with another object or given to another character. For further details about the <e-Adventure> language, refer to Moreno-Ger *et al.* (2007).

3.3 Game state

Just describing the elements that form the game yields a plain game where every door is always open, every character always says the same things and every exit leads to the same place. For the game to be interesting, it is necessary to support the means to provide a sense of narration. We can achieve this by introducing a notion of 'state'. All the actions that we perform in the game should be able to affect future actions. Some objects may be hidden until something happens (*e.g.*, the object appears only if the player has performed action X); some exits may be locked (*e.g.*, you cannot enter the library until you are admitted to the course or until you talk the secretary into letting you in); and a character may offer a different conversation (*e.g.*, the secretary is more friendly after the player gives her a small gift).

From the perspective of the author, these interactions are conceptually modelled by allowing each interaction (with an object or character) to activate conditions or, in <e-Adventure> terminology, flags. Then, the author can add preconditions to any element of the game. Intuitively, the state at any given point of the game is the set of active flags, which are an indication of which relevant actions have already been performed. See Figure 6 in Section 5.2 for an example of using flags in an <e-Adventure> document.

4 The communication between <e-Adventure> and IMS LD

The use of <e-Adventure> as an authoring environment for eGames addresses the issues related to the authoring of the games, but does not solve the authoring concerns regarding the integration of the games in the learning flow. As described in Section 2, the integration of eGames (implemented with <e-Adventure> or any other authoring methodology) is a complex task and raises a number of authoring issues. The basic problem is that when a learner is interacting with a specific UoL, the specification demands that the environment keep a record of the state of a number of variables, called properties, that can be used to alter the path of the learning flow. Likewise, eGames are often analysed in terms of game states which may or may not be directly expressible in terms of IMS LD properties.

It is necessary to provide the means to communicate and to translate the information used within the UoL and the information used within the eGames, a task that in general terms would require a strong programming background.

However, <e-Adventure> supports a clear and narrow eGame model, in which information is stored and interpreted in a declarative fashion. The task of authoring the game is facilitated by the use of a domain-specific descriptive mark-up language that can be understood and applied without a programming background. The same ideas can be applied to the authoring of the information flow, thus allowing the nontechnical author to specify the communications that should take place between the eGame and the UoL in a declarative fashion. These specifications written by the author are interpreted by the communication dispatcher shown in Figure 2.

The rest of this section describes the documents that should be created by the author to specify the translations between properties in the UoL and game states.

4.1 Mapping UoL properties to <e-Adventure> game states

While authoring an adaptive eGame using <e-Adventure>, the game designer is required to implement that adaptation in terms of conditions over the state of the flags, since that is the mechanism used within <e-Adventure> to make conditional decisions at any point. Indeed, if the state of a number of flags is modified as indicated in Section 3.3, the game can exhibit a completely different behaviour. If defined carefully, these different behaviours can correspond to different adapted versions of the same game.

Likewise, the state of the learning process at the moment of launching the game relies on a number of properties defined in the UoL. Following this declarative authoring approach, the author can identify the relations between sets of properties and states in an XML file (see Figure 7 in Section 5.2 for an example). When the game is launched, the communication dispatcher depicted in Figure 2 uses this configuration file to translate the state of the properties within the UoL into an initial game state.

4.2 Mapping <e-Adventure> game states to UoL properties

Once the eGame has been designed and written using <e-Adventure>, instructors or learning designers can also prepare separate documents identifying those game states that are relevant from a pedagogical perspective and that should affect the state of the current UoL. Again, the problem lies in translating <e-Adventure> game states into states of the UoL. As in the previous subsection, we use a declarative approach to allow the instructor to identify the relations between states and sets of properties in another XML file, with a mark-up syntax which is an extension of that used in the internal assessment engine implemented by <e-Adventure> and described in Martinez-Ortiz *et al.* (2006). In Figure 8 of Section 5.2 we include an example.

Each entry in this file is a mapping between a game state and a set of values for some of the properties present in the UoL. The game state is represented as a Boolean expression on the flags as used in the <e-Adventure> language itself (see Section 3.3). Meanwhile, the properties in the UoL that need to be modified are expressed with a list of set-property elements identifying the property to be set and its new value.

Given the nature of this process, it is important to note that the separation between this mechanism and the definition of the game in terms of states conditioned by flags is wide enough to take an authoring approach in which the writer of the game and the