Reusable and Usable Environment for the Digital Courseware Domain

Elsabé Cloete & Paula Kotzé
University of South Africa (UNISA), Pretoria, South Africa

cloete@unisa.ac.za kotzep@unisa.ac.za

Abstract

This paper considers a functional framework that creates a usable authoring support environment (ASE) for digital course design, and outputs reusable components. Within the context of considering the courseware domain as a domain of interactive software systems, we developed an ASE prototype. The objectives of this prototype include the provision of a usable authoring tool to develop interactive courseware, as well as the creation of domain products that are based on open standards to foster large-scale reuse of these products. In this paper we describe the software architecture of the prototype, based on usability requirements.

Keywords: domain modelling, e-learning, reuse, usability

Introduction

Electronic learning (e-learning) is a combination of learning services, technologies and products that provides a coherent institutional environment for instruction on the Internet. At the moment, most institutions with e-learning programmes have small-scale deployments of fully interactive digital courses, and tend to depend on learning management systems to create a sense of an e-learning environment. Since lecturers are responsible for creating learning opportunities, they are often also burdened with the task of being responsible for interactive digital courses. In some instances, institutions employ educational technologists to assist in this task. However, if the small-scale deployments ever become full-scale deployments, it is unlikely that there will be enough educational technologists available to develop all courses as interactive software systems. In an information technology environment, one would never expect non-experts to write complicated software programs, but in the teaching environment, we expect teachers to create learning opportunities within a framework that requires complicated software programs. Although some teachers may be inspired to create learning opportunities using all the technical tools at hand, few are actually able to create dynamic learning software that uses the tools in such a way that they blend into the background. Furthermore, maintenance of interactive course software is highly demanding. Content objects that are not designed according to open standards thwart the reuse of these learning objects to a large extent, and also prohibit the integration of content objects into central repository environments.

This paper describes the development of a domain prototype for the courseware domain. The objectives of the domain prototype are (1) to provide a usable authoring tool that lecturers can use to develop interactive courseware, and (2) to create domain products that are based on open standards to foster large-scale reuse of the products. The next section describes the courseware domain as an interactive system environment, and distinguishes between families of systems in the domain to clarify the

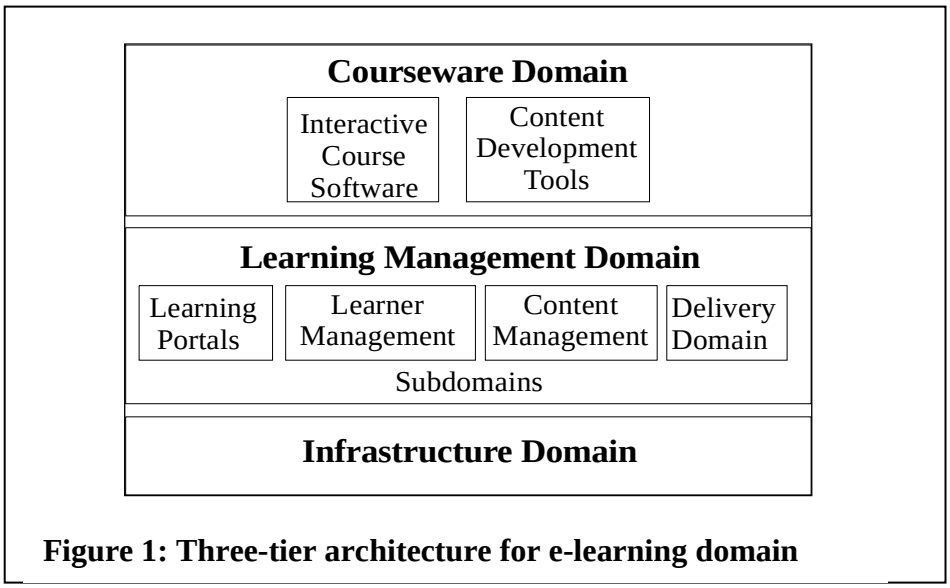

Figure 1: Three-tier architecture for e-learning domain

context of the research. The need for an architecture of the domain prototype is described in Section 3. Section 4 describes architecture and design considerations of the prototype. Our conclusions, in the final section of the paper, round off the paper.

Courseware Domain as an Interactive System Environment

The domains within e-learning are not always well understood, which means that institutions or individuals embarking on e-learning are often confused by the array of products available in the field. The term *domain* is used to 'denote or group a set of systems, or functional areas, within systems, that exhibit similar functionality' (Kean, 1998).

The domains within the field of e-learning can be organised in a three-tier architecture (Cloete, 2000) as described in Figure 1. The *infrastructure domain* resides on the bottom layer. Target systems in this domain provide networking and connection services for higher layers, which include activities such as providing for the transparent transmission of network messages and defining infrastructure specifications of hardware for domain users. Owing to the obviously technical nature of this layer, the uniqueness of the domain is not in doubt. The distinction between the next two layers is not always clear, which results in impossible expectations from target systems in the distinct domains.

The middle layer describes the *learning management domain*. The purpose of this domain is to provide middleware services by using the underlying infrastructure services to create and facilitate a reliable and effective learning environment. Target systems in this domain are called the *learning management systems* (LMSs) and they are responsible for enabling delivery, management and administration of the learning environment to take place. The learning management domain can be subdivided into four subdomains including the:

- learning portal subdomain
- learner management subdomain
- content management subdomain
- delivery subdomain.

The reason for this subdivision is that in some of the subdomains, separate off-the-shelf target systems are available that can be purchased and plugged into the learning management domain.

Families of *learning portal* systems provide an aggregation of learning services. These services are presented as a single access-point, via the web-browser. The learning portal is commonly embedded in the LMS where it performs functions such as maintaining course catalogues, news forums, learner collaboration forums, instructional support, etc. The *learner management* subdomain is responsible for activities such tracking and managing the learner competencies, facilitating and managing student registration, providing a centralised tracking system that monitors students performances and progress, reporting functions on a comprehensive set of views concerning learning programmes and learning progress, automatic notification triggered by specific events, et cetera. The primary goal of the *content management* subdomain is to manage the content objects that are stored in a content repository through activities such as indexing and retrieval of content objects, as well as implementing security measures that protect content, ensure customer privacy and shield student data from unauthorised access. LMSs essentially provide an environment in which the courseware resides and from where it can be *delivered*, which means that the delivery mechanisms for e-learning are usually embedded in an LMS. In the same way as for the other subdomains, separate delivery target systems are available for plug-ins. Examples of LMSs include WebCT, BlackBoard, LotusLearning Notes, ThinQ, et cetera.

The *courseware domain* resides at the top layer of the architecture. This domain is subdivided into two subdomains:

- the interactive courseware systems (the content itself)
- the content development tools that enable courseware designers to create the desired interactive courseware systems.

Content development tools provide authoring support environments (ASE) for developers to produce customised e-learning content for deployment on CD, the Intranet or the Internet. From the viewpoint of the learner, digital courseware conforming to the requirement of enhancing the learning process must allow its users to communicate interactively with the course software. Content development tools can broadly be categorised in two classes, namely the *programming class* tools and the *non-programming class* tools. Tools from the programming class, such as Toolbook II and Quest, are mostly products from ex-CBT vendors who have historically delivered CD-ROM-based solutions. The use of these tools requires expert technologists or programmers who grasp the basics of programming environments. Tools in the non-programming class are usually less complex to manipulate and can potentially be used by non-technologists to create interactive web courseware. Examples of these tools include products such as Front Page, Glorion, Adobe's Web Collection, etc.

'Interactive courseware' constructs a class of interactive software. This implies that the design and development of interactive course software require the use of software engineering principles to guarantee robustness and usability at least. However, most lecturers are not familiar with software engineering methodologies and even if they can manage the basics of a non-programming content development tool, the robustness and usability of the final product may be problematic. The fact is that their creators, or the institutions that 'sell' them, seldom regard digital courses as software, and most probably, if they did, few would attempt to experiment with e-learning. Although some institutions use education technologists to develop courses, many of these courses are in fact designed and developed by their teachers, of whom only a selected few understand or appreciate the underlying software architectures.

The Need for a System Architecture

The purpose of this section is to consider the architectural requirements for our domain prototype by referring to the context of software architectures, and the models that create a collection of multi-tiered abstract objects, together forming a consensus view of the software system.

During the design of a software system, several views are constructed to provide a set of solution architectural elements[1] to which all aspects of the problem domain are mapped. These architectural elements often include the structural model, framework model, dynamic model and process model. The *structural model* shows the structural decomposition of the system, depicting system components, their properties, configurations, constraints, semantics and rationale. In the *framework model*, the primary emphasis falls on the coherent structure of the whole system in its specific domain, rather on than the compositional structure of its problem classes. The *dynamic model* describes the dynamics involved in the progress of system computation and includes a description of the overall behaviour of the system, the behaviour of its structural elements, as well as their relations and collaboration with each other. The *process model* of the system, at a high level, identifies priority operations of the system and, at a detailed level, describes the procedures or process involved in those operations. Combined, the architectural elements compose the architecture of the software system and can be described as a topological organisation of a set of modules, their arrangements, interactions and interdependencies, together with a set of rules governing these modules to provide the system solution according to the requirements of the system.

The purpose of a software architecture is to serve as a basis for analysis and decision-making in order to produce consistent, integrated and usable system components over the design life cycle of the proposed software system. At the same time this architecture serves to identify tools, methods and facilities needed to develop the system solution (in our case a prototype of the solution) (Emery et al., 1996).

Prototyping refers to a way of presenting a minimal subset of the application's functionality before an extensive design and development life cycle begins. *Low-fidelity prototypes* are simple layout sketches or presentations focussing on content and layout in order to elicit user feedback as early as possible. *High-fidelity prototypes* are more complicated presentations (usually involving coding) that attempt to address the baseline functionality of the problem domain, including fully interactive capabilities that enable designers and users to analyse the flow between tasks.

System requirements are often classified as functional, non-functional or constraint requirements. The *functional requirements* refer to those requirements representing the functionality of the system from an external point of view, while the *non-functional* requirements describe the external behaviour of the system in terms of parameters to quantify the functional requirements, for example as performances, capacities, et cetera. *Constraints* are the conditions that set the system boundaries.

On an architectural level, Booch (1994) uses a systematic approach to present a more finely grained classification of requirements that are architecturally significant to a system. These include functional requirements, non-functional requirements, design requirements, implementation requirements, interface requirements and physical requirements.

To respond to the purpose of our prototype, namely to serve as a vehicle for the exploration of learning design, as well as to substantiate the contention that it is not necessary to assault teachers with technical coding, it was necessary to design and develop a high-fidelity prototype that includes a minimal subset of architectural requirements. We selected the following architectural requirements to design our prototype:

- Functional requirements: to include capabilities for creating or updating a new course and exporting the domain products of that course to an open standard.
- Non-functional requirements: to ensure usability and robustness of the system as well as reusability of its products.
- Design requirements: to take into consideration system constraints within which a desired solution system (based on the prototype) should exist.

[1] The architectural elements are also commonly referred to as ‘views’ or ‘models’

Functional Requirements

Richter (1999) suggested an approach to obtain the functional requirements of the system at hand, by defining the following:

- The designer/user defined overall goal of the system - in other words, what must the system achieve?
- Who are the external role-playing agents in the system?
- In what goal functions are the external agents involved?

In our case, the overall goal of the system is to present a user with a usable interface allowing him or her to create a new learning opportunity, or update an existing one, and to export the results to an XML-based (EML[2]) system. The user (as course developer) is the only external agent in the system. The external agent will be involved in the following functions:

Primary actions:

- creation of new learning opportunities
- export of learning opportunities (for reuse of domain products).

Secondary actions:

- recall and update of existing learning opportunities
- provision of a complete text view of all course components included in the course
- deletion of an existing course.

Design requirements

The focus of our design is to create a proof-of-concept prototype. Because of limited resources and the decision on proof-of-concept, we decided to focus on the innovative aspects of the problem space where we demonstrate that it is indeed possible to hide the implementation details of standards from the user. We therefore created an ASE tool with limited functionality that does not reflect every aspect of interactive learning, but serves instead as a domain model that could be used as an XML-wrapped content object generator.

However, we had to work within certain design constraints: Pending negotiations between our institutions and the creators of EML (Hermans et. al., 2000), we had to use the EML DTD as the underlying standard and could not create our own XML tags.

Our prototype focuses on the *information component subsystem* of the courseware domain (see the next section). The final product of this subsystem is a simple e-learning interface that provides course information, not content information, to the user (learner). The course information includes aspects such as course objectives, prerequisites, syllabus, discussions on how the learner's progress is assessed, course material (such as textbooks, videos, audio, etc.), as well as all the assignments of the course. The prototype produces a reusable EML product file that is of no use to the end user (learner). However, in the next subsystem of the courseware domain, this EML product serves as input to the *transformation subsystem* where it is transformed into the resulting e-learning system that the user (learner) can use (represented by the mapping between the resulting e-learning system and the e-learning interface, as illustrated in Figure 2).

[2] EML refers to the educational modelling standard.

Non-functional requirements

Our defined functional requirements and subsequent analysis approach affect the usability of the system. *Usability* is concerned with interface aesthetics and consistency, and also forms an integral part of the tasks, intentions, goal and models of the intended system. We describe these terms in the context of our problem space.

Tasks are operations to manipulate the concepts of a domain, while a *goal* is the desired output from a performed task. A goal requires an *intention*, which is a specific action to meet the objective of the goal. *Task analysis* refers to the identification of the problem space for the user of an interactive system, in terms of domain, goals, intentions and tasks (Dix, 1998). Two general approaches to task analysis exist, namely a *user-oriented approach* and a *system-oriented approach*. In the user-oriented approach, the intention behind task analysis is to capture what has to be known about using a system in order to achieve goals. Some of these activities may involve aspects that are not, and are not expected to be, part of the software system under design or development.

In contrast, the system-oriented approach can be defined in terms of accesses to the functionality of the interactive system in order to determine or modify its state. According to this approach, tasks depend on the part of the interactive system to which they refer (Paternó & Leonardi, 1994):

- *Presentation-related* tasks do not require access to the functional core of the application and refer to the set of functionalities that are independent of the interaction objects used to interact with the user. This includes tasks such as selecting, resizing or scrolling windows, selecting visualised objects, customising dialogue boxes, etc.
- *Application-related* tasks require access to the functional core of the application and refer to parts of the interactive system that are independent of the general components used to interact with the user. In our domain, this refers to the general functionality of the authoring support environments.

The first of these aspects relates to the appearance of interaction objects, while the second relates to the application semantics. In this paper the focus is on the latter. Focussing on interaction objects in terms of their semantic effects does not mean that appearance-related aspects are not all that relevant. Inappropriate use of windowing techniques, scrolling and colours can result in tedious and confusing interaction with the computer. Detailed domain knowledge in the design of these is thus just as important (Gulliksen, 1995).

Generally, an ASE can be defined as consisting of a number of goal-directed processes and a number of events driving these processes. The goal of the processes and events is to produce an operational, interactive e-learning system.

A fundamental model of interactive behaviour requires states and commands that transform these states (Dix, 1998). The abstract model of the system we are going to use consists of a set of states and relationships between these states, as well as certain operations that are allowed on these states. In the courseware domain there are four state subsets of interests when interacting with an ASE, as illustrated in Figure 2 (Kotzé, 1997):

- The *internal ASE state* or *system state* refers to the functional state of the ASE, including aspects such as the internal state of the e-learning objects under development as represented within the system (what the 'recorded' system state of the product under development is), as well as those relating to the general functionality of the ASE being used (the variables needed to control the interaction with the author, for example what the current object being edited is, the templates and the default values used for specifying attribute set values, the format of the prompt lines, etc.)
- The *ASE interface state* represents the external displays of the ASE while authoring. Also called the *display state* of an ASE, this state refers to the external perceivable rendering of the internal

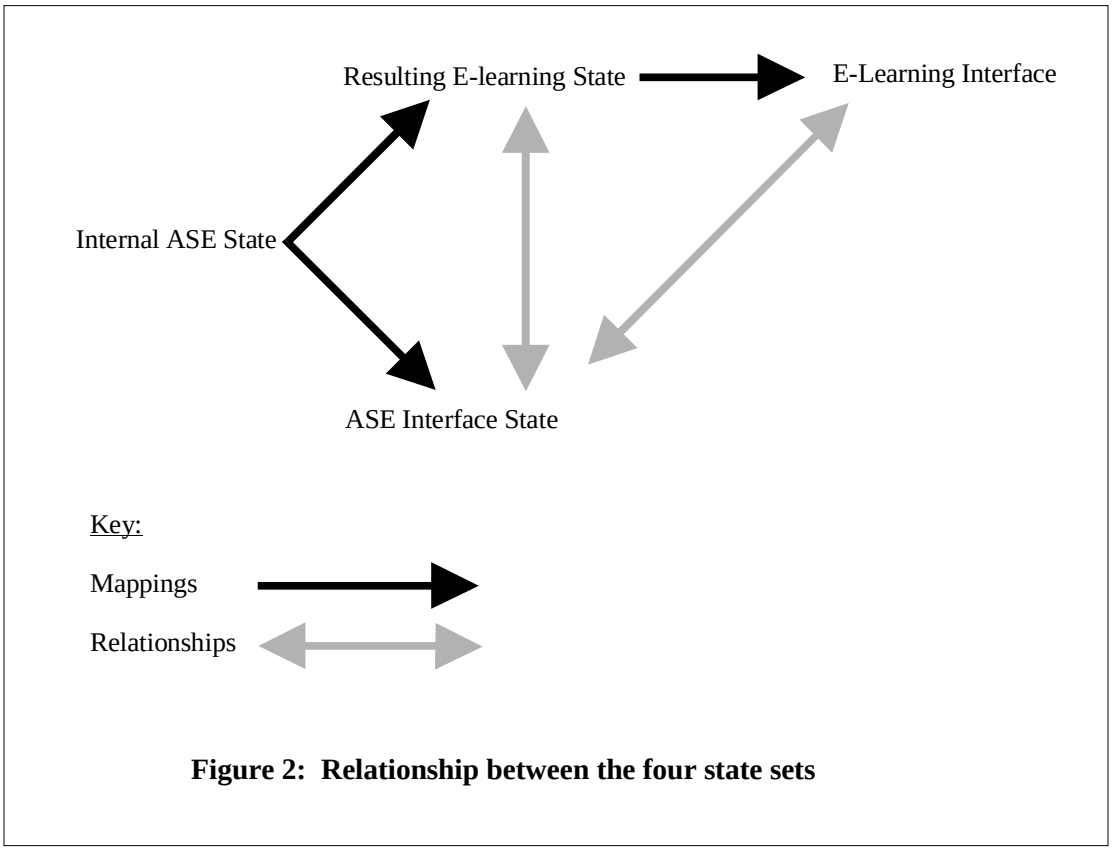

Figure 2: Relationship between the four state sets

state as reflected at the ASE interface, which is the 'perceived' state of the product under development.

- The *resulting e-learning state* characterises the resulting e-learning system and refers to the actual state of the e-learning program under development - the program that will determine what the student will get in the delivered product and the rules for presentation.
- The *e-learning interface state* represents the external display of the e-learning system and refers to what the student will perceive at the interface of the e-learning system - a student will perceive instructional content, but not the rules governing the sequence in which the instructional content will be displayed to him or her.

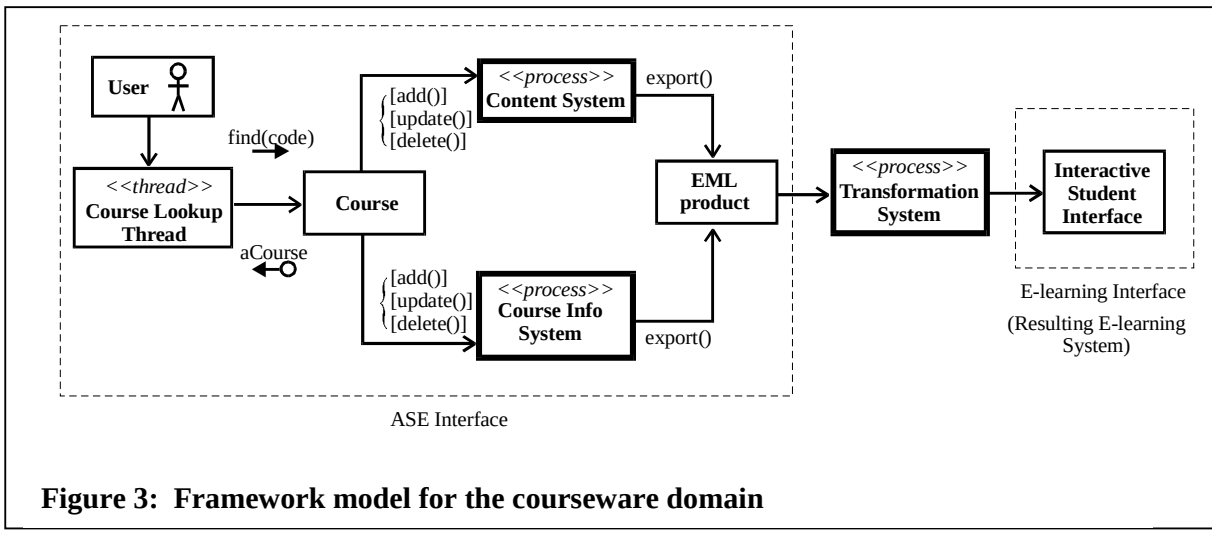

Figure 3: Framework model for the courseware domain

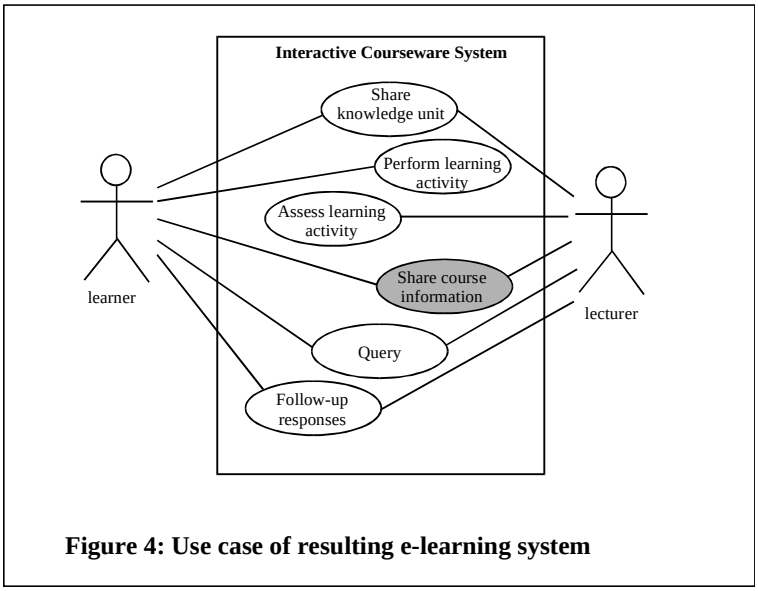

Figure 4: Use case of resulting e-learning system

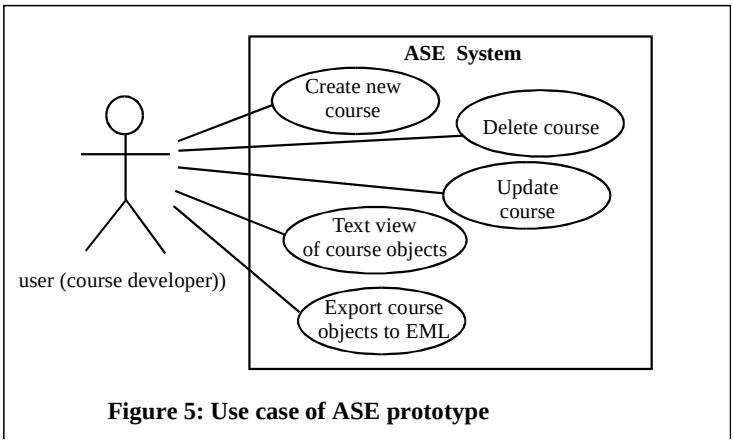

Figure 5: Use case of ASE prototype

Our main interest lies in the relationship between the external ASE display and the e-learning result (how the resulting e-learning system is reflected at the ASE interface during the authoring process). For the prototype we are not interested in the relationship between the external ASE display and the e-learning student display (how the student will perceive the resulting e-learning system is reflected at the ASE interface during the authoring process). This decision becomes apparent when we describe the architectural framework of the system.

Considering our focus for this prototype, we are mainly interested in the usability principles relating to the robustness of the system. The *robustness* of a system refers to the features which support the successful achievement and assessment of the goals, and include issues such as observability, recoverability, responsiveness and task conformance (Dix, 1998). The focus of the prototype lies in both *observability* and *task conformance. Observability* refers to the ability of the user to evaluate the internal state of the system by means of the perceivable attribute representation of the system at the interface (Dix, 1998; Kotzé, 1997), i.e. the relationship between the Internal ASE state and the resulting e-learning state as reflected at the ASE interface. Throughout the authoring process the author should be able to observe exactly what effect his or her actions would have on the delivered product. As described before, the prototype results in an EML product file that is not the final interactive e-learning system with which the students and lecturers interact. Our observability interest is therefore not to observe the exact effects on the e-learning system, but rather to observe what components were included. This aspect is portrayed in the 'Text View' of the information component subsystem. *Task conformance* or compatibility refers to the degree to which the system supports all of the tasks of interest (referred to as 'task completeness') and whether it performs these tasks to the users liking (referred to as 'task adequacy').

An architectural view of a usable and reusable courseware domain

According to Figure 3, the framework model of the courseware domain consists of three subsystems, namely the *content subsystem*, the *course information* subsystem and the *transformation subsystem*. The objectives of the *course information* subsystem are to provide the user (lecturer) with an ASE tool that assists him or her in entering general course information such as the course objectives, prerequisites, syllabus, assignments, et cetera. The *content information* subsystem, on the other hand, provides the user with an ASE tool for entering interactive learning units and content knowledge objects, in order to

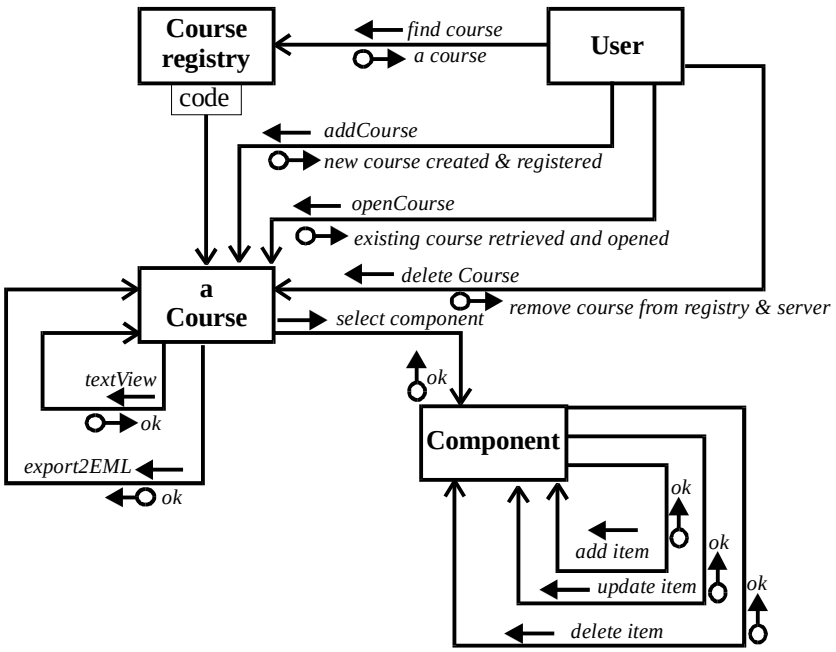

Figure 6: Activity diagram for the ASE tool illustrating the flow of the system

compose a specific course or learning opportunity. Both these subsystems output an EML file that either consists of a single course, or different course components that may be reused when developing other courses, either by inclusion of these components, or merely by reference to them. The *transformation* subsystem uses the output from the content and course information subsystems as input to produce a usable e-learning interface through which end-users (lecturers and learners) can interact. The focus of our domain prototype is only on the course information subsystem and how the user interacts with it.

Before modelling the system functions suitable for the ASE prototype, we briefly consider the functional model of the resulting e-learning system so that it becomes clear where we are heading and exactly how the ASE domain prototype fits into the courseware domain. Figure 4 describes how the actors (learners and lecturers) will be using the resulting e-learning system. During a learning opportunity, learners and lecturers interact with the interactive courseware system by sharing course information, knowledge units, performing learning activities, asking questions (queries) and responding, performing follow-up activities. Lecturers also assess learner progress. We focussed our domain prototype on only one of these activities, namely the sharing of course information (the shaded use case). This focus is refined in Figure 5, illustrating the actors and subsystems involved in the process of sharing course information. Two primary actions are identified for the prototype, namely (1) the creation of a new course, and (2) the export of created course components to EML. To increase the usability of the prototype three more actions were selected for inclusion, namely (3) the updating of an existing course, (4) the ability to view a set of all components that were selected for inclusion for a particular course, and (5) the removal of a course.

Figure 6 illustrates a dynamic overview of the ASE prototype by describing the temporal behaviour of the system. According to this dynamic model, the user must be able to either create or access a specific course. The design must therefore maintain a list of courses, each of which is identified by a course identifier (course code). A course registry can be used to look up each course instance using the course identifier as qualifier. The association between the course class and the course registry is clearly illustrated in Figure 6. As illustrated, the main actions available to the user are as follows. The user uses the qualifier to retrieve an existing course for update or removal, or to create a new course. The prototype follows an

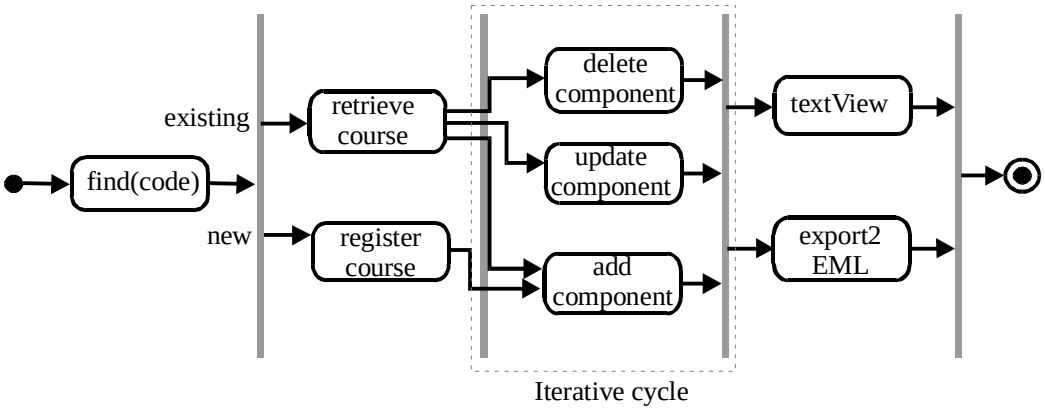

Figure 7: Activity diagram for the ASE tool illustrating the flow of the system

approach by which it guides the user by making available a set of course components that can be included into a single course. Because we are constrained by the current DTD elements of the EML standard, the number of components is also restricted. We selected only a sensible subset of all components, available for information sharing, for inclusion in the prototype. Each course eventually comprises of a set of course components selected by the user. The combination of components may be unique per course. When the user selects a specific course component, depending on the component, the user can perform a number of functions on that component such as adding new instances, of that component, updating specific instances, deleting specific instances, and paging through the instances. Figure 7 provides a different view of the dynamic model of the system through activity diagrams describing the flow of the system.

Figure 8 depicts the static view of the complete functional model of the ASE prototype. The different course components with the attributes and expected methods are clearly indicated. For the sake of completeness we use the assignment component to illustrate some of the details of the system.

Figure 9 illustrates the dynamic model of the system when considering interactions with the assignment component. According to the business rules of the system, the user must include all assignments that a learner must complete and submit throughout the course, in the component information subsystem. Each assignment has attributes such as a description of the assignment environment, specific objectives for the assignment, specific prerequisites for the assignment, the due date by which the assignment must be submitted for assessment, et cetera. Furthermore, an assignment is comprised of questions, and we distinguish between multiple choice questions and other questions. The purpose of the multiple choice questions is to allow for questions that are multiple choice in the true sense of the word, or true/false questions, or questions that may have more than one correct answer. Three iterative sessions are defined within the assignment component. The inner iteration materialises in instances where a single question may have more than one correct (or incorrect, in the case of multiple choice questions) answer. The second iteration occurs where more than one question may be defined within a specific assignment, and the outer iteration takes place where a course can have several assignments. Within each of these iterations the basic operations include add, update, delete and paging through the component items (answers or questions or assignments). Figure 10 shows a screen shot of the assignment environment within the ASE prototype.

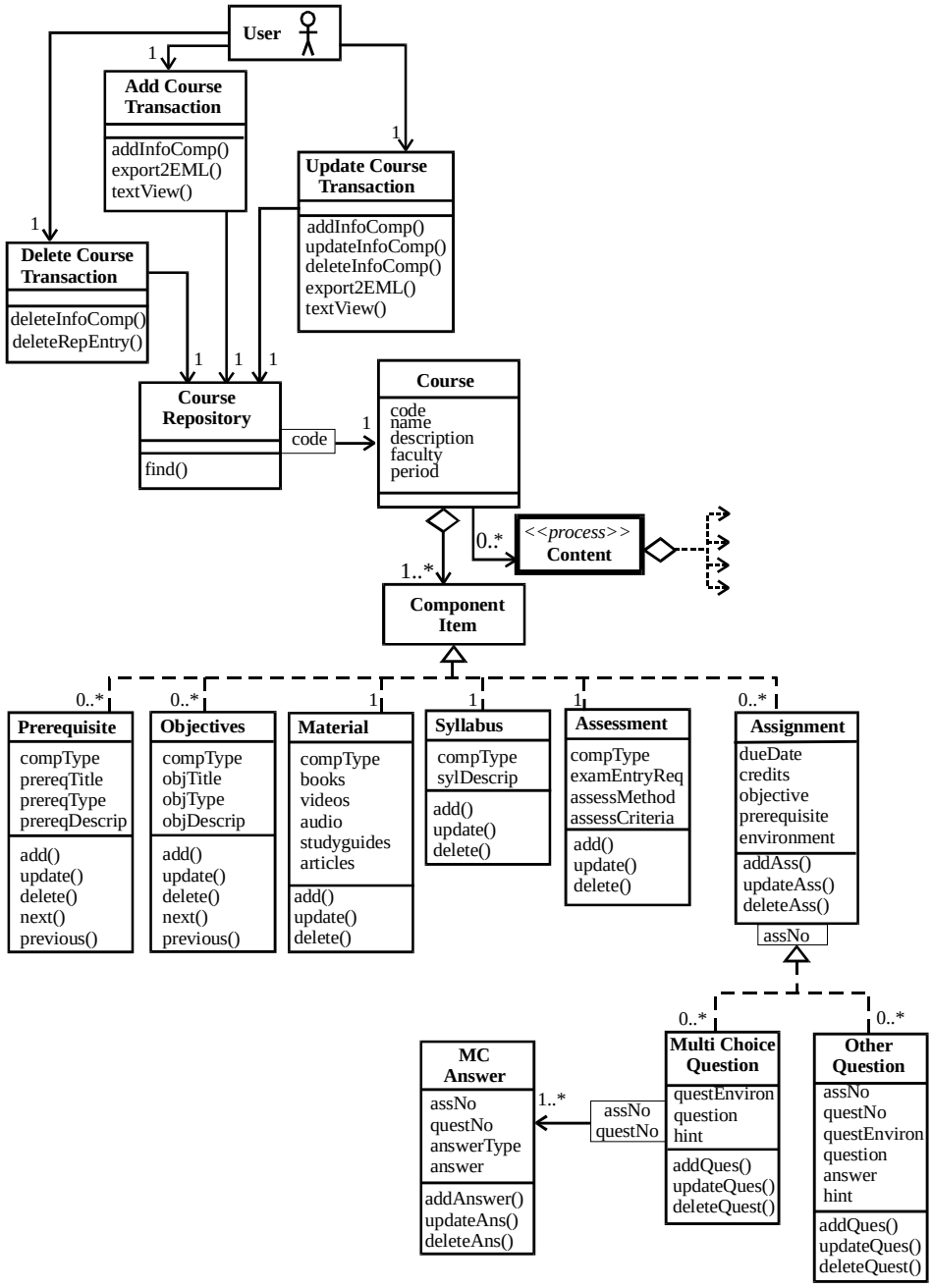

Figure 8: Structural model of the complete component information subsystem

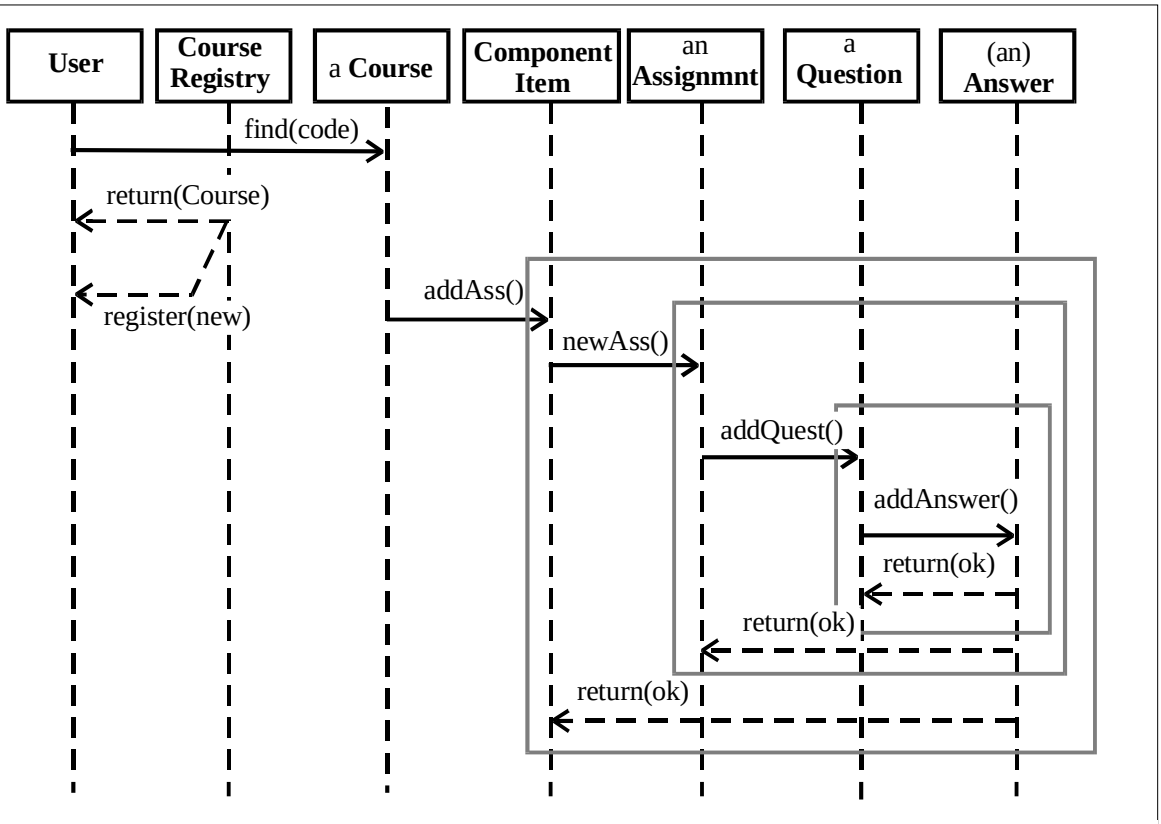

Figure 9: Dynamic model of the system when considering interactions with the assigment component

We conclude our discussion on the ASE prototype by briefly considering the activities of the export transaction. The export transaction interacts with an EML repository containing one-to-one mappings between the components and their corresponding EML tags. Each component has at least two tags associated with it, namely a startup-tag and a stop-tag. Furthermore, each component can also be comprised of an unspecified number of items. These items may be iterative. During the export transaction, each component iterates through a method of:

- Finding its own startup-tag from the repository and pre-affixing the tag to itself.
- Placing its associated stop-tag on top of the stack.
- Scanning for any items within it. For each nested item it finds, the procedure of pre-affixing the startup tag to the item and placing the associated stop-tag on top of the stack is repeated.
- When a new (non-nested) item is encountered, the stop-tag on top of the stack is removed and appended to the last item. The scanning procedure is then repeated.
- When no more items for a specific component are available, the stack is emptied from top to bottom, appending each of the stop-tags still on the tag to the (now) EML-object.

Summary and Conclusions

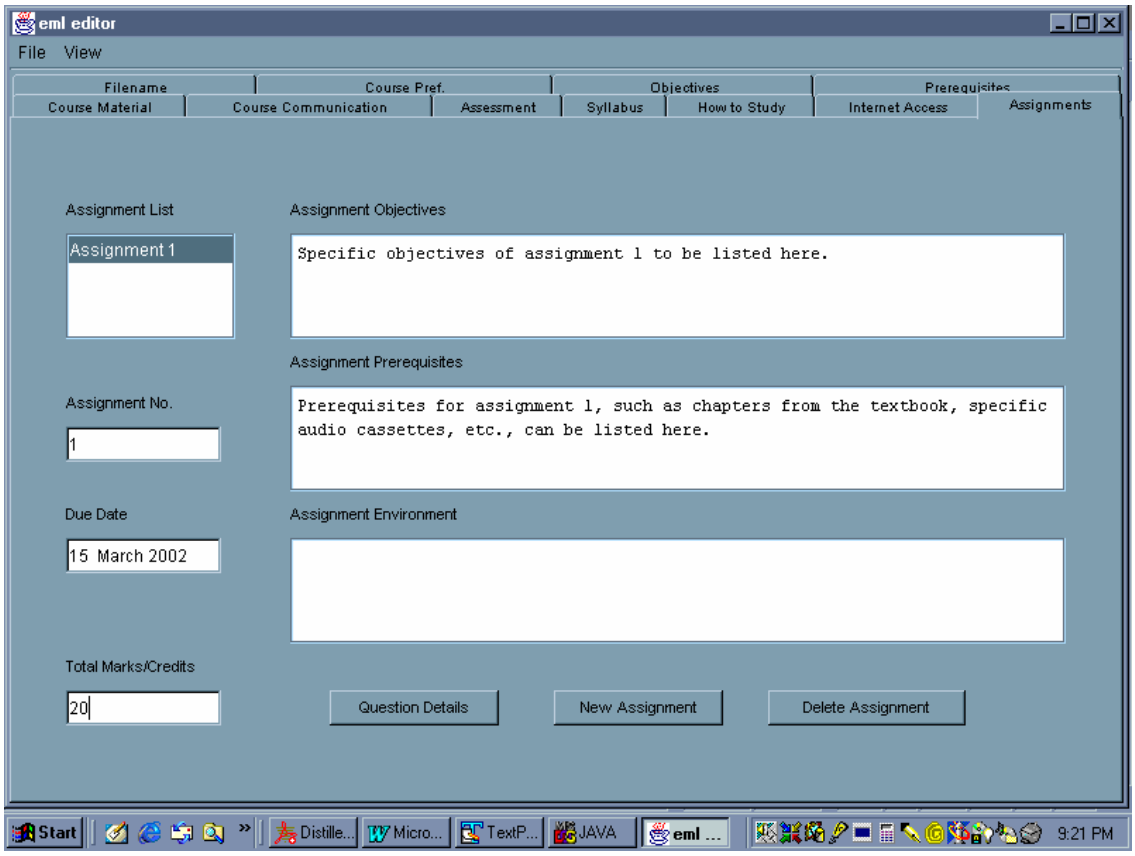

Figure 10: Screen shot from ASE prototype in the *assignments* environment

The courseware domain is organised into a three-tier architecture comprised of three subdomains, namely the infrastructure domain, the learning management domain and the courseware domain. The paper focussed on the latter - a functional framework within the courseware domain that creates a usable authoring support environment (ASE) for digital course design and outputs reusable components. Our ASE domain prototype presents a simple way to construct learning objects and export them to an open standard for reuse without burdening the user with the technical details of the required standard. In continued research, we are extending our framework, building more prototype components to fit the other subsystems within the courseware domain.

References

Booch, G. (1994). Object Oriented Analysis and Design with Applications. Addison Wesley.

Cloete, E. (2001). Electronic Education System Model. *Computers & Education*, 36 (2), pp. 171-182.

Dix, A., Finley, J., Abowd, G., & Beale, R. (1998). *Human Computer Interaction. 2nd Ed.* Hemel Hempstead. Prentice Hall.

Emery, D.E., Hilliard II, R.F., Rice, T.B. (1996.) Experiences Applying a Practical Architectural Method. *Retrieved in November 2001 from the World Wide Web:* http://www.adahome.com/Resources/Papers/General/Archi-EmHiRi.html

Gulliksen, J., Sandblad, B. (1995). Domain-specific design of user interfaces. *International Journal of Human Computer Interaction.* Vol 7 (2). pp 131-151.

Hermans H.J.H., Koper E.J.R., Loeffen A., Manderveld J.M., Rusman E.M. (2000). *Reference Manual for Edubox-EML/XML binding 1.0/1.0 (Beta version).* Onderwijstechnolgisch expertisecentrum OTEC, Open Universiteit Nederland. Website: http://www.edubox.ou.nl

Kean, L. 1998). Domain Engineering and Domain Analysis *Retrieved in November 2001 from the World Wide Web:* http://www.sei.cmu.edu/str/descriptions/deda_body.html

Kotzé, P. (1997). The Use of Formal Methods in the Design of Interactive Authoring Support Environments. *PhD Thesis*. Department of Computer Science. University of York (UK).

Paternó, F., Leonardi, A. (1994). A semantics-based approach for the design and implementation of interaction objects. *Computer Graphics Forum.* Vol 13 (3). pp. C-195 - C204.

Richter, C. (1999) *Designing Flexible Object-Oriented Systems with UML*. Indianapolis, USA. MacMillan Technical Publishing.