Systematic Development of Courseware Systems

Claus Pahl
Dublin City University
School of Computing
Dublin 9, Ireland
cpahl@computing.dcu.ie

Abstract: Various difficulties have been reported in relation to the development of courseware systems. A central problem is to address the needs of not only the learner, but also instructor, developer, and other stakeholders, and to integrate these different needs. Another problem area is courseware architectures, to which much work has been dedicated recently. We present a systematic approach to courseware development – a methodology for courseware engineering – that addresses these problems. This methodology is rooted in the educational domain and is based on methods for software development in this context. We illustrate how this methodology can improve the quality of courseware systems and the development process.

Introduction

The development of teaching and learning environments has always been a challenge. Learning itself is a difficult subject, often not entirely understood. The needs of different groups of users, such as learners and instructors, but also other stakeholders that play a role in administration and management, are often difficult to determine and to integrate. The development of software systems for teaching and learning adds an additional complexity. *Educational technology* on one hand and *software technology* on the other are very distinct. Our objective here is to bring the educational domain and software development for courseware together.

Substantial progress in computing technology has boosted computer-supported teaching and learning. A variety of *courseware systems* are widely and successfully used. This makes it now an appropriate time to investigate a systematic way of their *development* deeply rooted in the educational context that will address the problems that still remain in the area. Some effort has already been made in this direction. (Boyle, 2003) focuses on the reusability of learning objects. (Pantano Rokou, Rokou & Rokos, 2004) address the aspect of pedagogical modelling in this context. Our view is an architectural and behavioural one on courseware systems that complements the other efforts. We understand these as systems composed of components that interact in order to support learning processes. We present our methodology from a development perspective by identifying separate stages.

The integration of all groups involved in courseware development and their active participation throughout a courseware lifecycle is paramount (Virvou & Tsiriga, 2001). In addition to a participative style of development, we found in particular an openness and flexibility of the courseware architecture important. Participation and the architectural design are, therefore, the cornerstones of our methodology, tailored towards the needs of educational technology and courseware systems engineering. This methodology is rooted in the educational domain through an information model, based on different facets of courseware systems.

Courseware Systems Facets

Courses are complete units of instruction. *Courseware* is software intended to train or instruct based on a course. A *courseware system* is the combination of course content and courseware (IEEE LTSC, 2001). Courseware systems development is a complex activity. It is more than instructional design; it involves learner and instructor, but also the organisation offering a course and the developer providing necessary software technology. In order to capture all aspects, we introduce a *facet-based courseware description and classification model* (Pahl, 2003). The objective of this model is to provide a facetted description and classification scheme for courseware systems reflecting the different perspectives and perceptions of different stakeholders.

- *Content* – the subject-oriented perspective. The learner works with the course content for a subject, which is, however, created by the instructor.
- *Format* – the organisational perspective. The organisation determines aspects such as syllabus, stakeholders, or aspects of the environment.
- *Infrastructure* – the technical perspective. Both organisation and developer are responsible for determining the infrastructure technology.
- *Pedagogy* – the educational perspective. The instructor is the key factor in determining the pedagogy.

The facet model can serve several purposes: to classify existing courseware systems and to guide the requirements organisation in a courseware development process. We can use the facet model to classify our case study system – an undergraduate introduction to databases for a computing degree, called IDLE (Interactive Database Learning Environment).

- *Content*. Content is represented using HTML and XML for text-based material and audio, flash animations, and Java applet and servlet technology for active learning elements. The educational services that are offered are lectures, tutorials, labs, and self-assessment. The system is mainly a delivery system; only a simple interface for the XML-based course representation supports authoring.
- *Format*. Stakeholders in our system are the lecturer, software developers, the university, and students. The syllabus focuses on an integration of theory and practice, i.e. database languages, mathematical theories, and design methods. Standards of importance include XML-based educational markup, learning environment architectures, and Web-related standards. Intellectual property rights – part of the legal environment – affect the university (infrastructure) and the lecturer (content).
- *Pedagogy*. A number of educational activities including autonomous learning and active learning are supported. These activities require the support of learning processes, interactivity among various system components, and an integration of the courseware system with a database system for practical and project work. Since this is a support environment for an on-campus course, only learner-content interaction is supported.
- *Infrastructure*. The system software comprises Web platform tools including multimedia plugins, a Java servlet architecture, and a database system. We deploy a stand-alone server as the hardware platform.

This classification could be considered as example of an initial *requirements specification* for a courseware system. To pursue this direction, the descriptions need to be detailed further.

Standardisation bodies in the area have introduced annotation and classification schemes. A detailed description model is the Learning Object Metadata Standard LOM (IEEE LTSC, 2002). The purpose of LOM is to support the management, location, evaluation, and development of learning objects – which covers a wider range of activities and a wider range of objects than our courseware systems context. We use the facet model as a classification tool and as part of our development methodology. LOM provides a categorisation of aspects similar to our facets. The LOM categories ‘technical’ and ‘educational’ overlap with our ‘infrastructure’ and ‘pedagogy’ facets, respectively. Not all LOM aspects are needed in our context; on the other hand, some additional aspects are needed to consider specific development aspects.

Courseware Systems Engineering

A Courseware Systems Engineering Methodology

Educational software is different from other software applications. It can be characterised by as user-centred and interoperable with other systems and standards – which reflects the key characteristics based on our experience.

- *User-centred systems*. In traditional forms of software development, the client and potential users are contacted at an early stage to elicit requirements. Often, however, very little contact remains until the final product is available, resulting in products that do not match the actual needs of the users. In user-centred software systems, there is now a recognised need for an ongoing participation of all user groups in the development process to clarify and validate existing requirements and to capture emerging ones (Bødker, 2000).
- *Interoperability*. Often, a course consists of different components – such as authoring system, delivery system, or evaluation system – that need to be integrated and work together. In most cases, a particular course will be part of a larger unit such as an undergraduate degree programme. In this case, the integration into a common

platform or interoperability with a student administration system are critical. Consequently, a need for interoperability and an open architecture arises (IEEE LTSC, 2001).

We introduce a development approach based on methods that cover different stages in the development process:

- *Participative design* is based on sociological principles and is an integral component in the field of human-computer interaction (Bødker, 2000).
- *Architectural design* is a classical software engineering technique to support the development and management of complex maintainable software systems (Bass, Clements & Kazman. 2003).

We propose an *incremental process model* that reflects the methodology requirements participative and open. The objective behind the facet model is to provide a general framework that considers the needs of all stakeholders. Usually, as a courseware development progresses, the focus shifts from mostly usage-oriented requirements to technical details of the implementation of the software and hardware system. The *development methodology* is based on languages and methods specific to the individual stages. Two *languages* are central – a scenario language and an architecture language. The *methods* are participative and architectural design. We will later on discuss how the facet model relates to the languages and methods.

Participatory Requirements Elicitation and Organisation

Participative design is a user-centred development approach, in particular suitable for software systems with a high degree of user interaction and complex processes involving the users (Bødker, 2000). The focus is on usability requirements. Central concepts of the approach are *scenarios*. Ideally, all stakeholders participate actively in the development process. Besides scenarios as static requirements representations, *executable prototypes* of the software system play an important role in the communication with the users. Software prototypes operationalise the scenario definitions (Beynon-Davies & Holmes, 2002).

Scenarios are constructions meant to stage activities and to reflect on and illustrate problems with these activities (Bødker, 2000). Scenarios abstract the user's current and future goals and tasks; they serve to predict the user's actions in the system. Scenarios are rooted in specific situations from the domain under scrutiny. A scenario serves as a medium of communication between users and developers. Alspaugh, Anton, Barnes and Mott (2001) define a *scenario* as a linear sequence of events, with associated attributes describing these events. An event is an association of an actor and an action. We refine this definition. We do not require a linear sequence. Instead, we allow a richer set of combinators reflecting the interaction processes of typical user-centred software systems where a user can choose between options, can repeat elements, or work on several elements at the same time. We also expand the notion of events, calling it an activity. An activity shall here be comprised of an actor, an action, and the object on which the action is carried out. This definition is an adaptation that serves to accommodate the complexity of the educational domain, in particular learning processes. We define the following *scenario language*:

- A *basic activity* is described as a triple (*actor*, *action*, *object*) consisting of an *actor* (a stakeholder), an *action* (part of the functionality of the system or an action affecting the system description or implementation itself), and an *object* (for example content or software).
- Simple activities can be composed to complex ones using *activity combinators*. We suggest *option* and *repetition*, which apply to a single activity, and *choice*, *concurrency*, and *sequence*, which can combine two or more activities.

We can distinguish two basic *types of scenarios*. *Direct scenarios* describe activities of users (usually learner and instructor) within the system. *Indirect scenarios* describe activities of stakeholders (usually administration and development staff) in relation to the development and management of the system itself. Two examples from our database courseware system IDLE shall illustrate scenarios. The first is a direct scenario describing a learning activity – exercising database queries – in an iterative cycle of a sequence of basic activities:

```
repeatedly
   sequence of
      (student, selects, query exercise)
      (student, reads,   query specification)
      (student, submits, query solution)
      (system,  replies, query result)
```

The second is an indirect scenario describing the maintenance of a query exercise database:

```
repeatedly
   chooses between
      (educator, adds,     query exercise)
      (educator, modifies, query exercise)
```

We have used scenarios as a communication medium in particular between instructor and developer (learners were more involved using prototypes), but scenarios also constitute specifications of activities of all stakeholders needed in the design and implementation process. The scenarios reflect the requirements and design issues of the different stakeholders focussing on their activities. The structure of scenarios is related to the structure of the facet model. Scenarios consist of activities based on subjects, actions, and objects. These objects are usually data objects or services, i.e. content in facet terms. Actions and activities are often learning process and interaction descriptions, i.e. pedagogic aspects represented in direct scenarios. Possible activity subjects are stakeholders (which are part of the courseware environment); i.e. subjects relate to the format in facet terms. Scenarios are linked to the implementation (the infrastructure facet) through prototypes (and an architecture mapping, as we will see later on). It has, however, to be noted that scenarios are not a direct reflection of facets, but rather that scenarios are rooted in the domain model described by the facets.

Architectural Design

Software architecture is a software engineering discipline that addresses the organisation of software systems into composable software entities (Bass, Clements & Kazman, 2003). A *software architecture* consists of components and connectors. *Components* are software units that provide a range of coherent services. In terms of the LTSA standard (IEEE LTSC, 2001) our components comprise processes and stores. *Connectors* are entities that represent connections and data flows between components. The main focus of software architecture design is the separation of computation and communication, which supports the maintenance of software. Software architecture bridges the gap between an initial concept of a system and low-level component implementation.

Scenarios can form the starting point for an *architecture definition*, supporting the transition from requirements engineering into the architectural design stage through a scenario-architecture mapping:

- Scenarios can be categorised according to their interactions with others. This might indicate implementation through the same component based on similar features addressed in the scenarios.
- Based on the scenario categorisation, the next task is to find an architectural style that structures component features and interactions. We suggest the LTSA as the basic architectural style – see Fig. 1.
- Required basic infrastructure services constrain the underlying platform and some aspects of the architecture: coordination of system activities, how data moves through the system, or integration of new components. Targeting the Web as the implementation platform is such a constraint.
- The next stage following the design is the operationalisation of components and connectors. Calling conventions, codings, and protocols need to be addressed through explicit bindings to software encodings such as APIs, data formats, and communications mechanisms, respectively (IEEE LTSC, 2001).

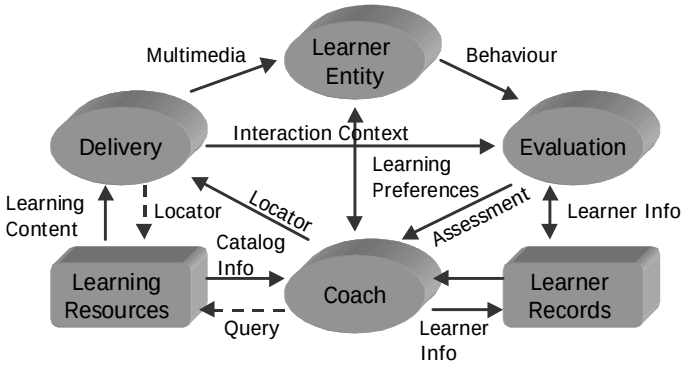

Figure 1. Learning Technology Systems Architecture LTSA.

The main elements of an architectural design are the components, their connections, and the component interaction processes. We illustrate these elements using our case study system IDLE.

- *Components*. The IDLE architecture consists of three subsystems: authoring, delivery, and evaluation. Within the subsystems, we identified component clusters; for instance, the component clusters in the delivery subsystem are the learner entity (through a user interface), the central delivery server itself, and the learning resources database – which implements the classical three-tiered architecture pattern for Web-based systems.
- *Connections*. Components are connected to each other in order to enable interactions. A connection enables the activation of a service at the other component and the flow of data between them. The component connections between the clusters in the delivery subsystems are presented in Fig. 2.
- *Interaction Processes*. Component interactions usually follow given protocols, for example for the SQL tutor, repeated interactions of the learner at the GUI with content in the resources database will take place. This type of interaction is an implementation of the learning process supported by the system.

We have choosen LTSA as the architectural reference model, which we have used here as a design tool. If interoperability of the executable system is required, other standards such as SCORM RTE (Runtime Environment), see (ADL, 2004), and other SCORM standards such as SN (Sequencing and Navigation) also have to be considered.

A *scenario-architecture mapping* is presented in Fig. 2. It describes the mapping of the SQL tutor scenario with the corresponding prototype onto the clusters of the delivery subsystem.

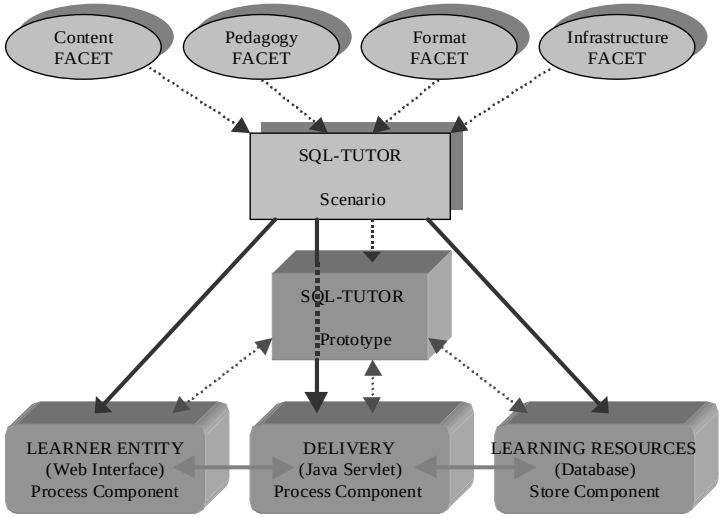

Figure 2. Scenario-architecture mapping.

Architecture descriptions are also rooted in the facet model. Data flow between components and services relates to the content facet. Component interaction patterns and services relate to pedagogy. External components and meta-level aspects relate to the format. The architecture platform is related to the infrastructure facet.

Discussion

Educational technology for computer-supported environments has reached a basic level of maturity – core principles and concepts are agreed upon. A wide range of courseware systems has been developed, based on educational technology principles and approaches. A *methodology for courseware engineering* can capture existing experience and to integrate further developments into this framework.

- The development of complex systems requires planning and a systematic approach. A methodology provides the languages, techniques, and methods needed to maximise the quality and to minimise the risk of problems typical for a domain. In our case, the lack of a systematic approach has caused difficulties in past.
- Specific aspects of the educational domain – e.g. usability and interoperability – require targeted solutions. We have integrated different aspect-specific methods such as participative development and software architecture.
- Using this methodology has been beneficial for the systems we have developed and has led to improved, effective system development processes.

The methodology – here illustrated for the IDLE system – has also been beneficial in the development of other systems that we have been involved in, including an adult literacy course, a synchronous learning platform for

distance education, and a courseware generation tool. Usability has been a key requirement for all systems; the second and third system are instructor-focussed. Interoperability and integration with other systems has been an important aspect for the second and third system, whose integration into IDLE is important. A methodology aims to improve the quality of a product and to reduce the risks involved in the development and management process. A methodology for courseware development and management needs to address the central features of the educational context. Let us note two observations reflecting the experience we, and others, have made. Firstly, *usability* is a unanimously agreed requirement of courseware systems and, secondly, the *architecture* is an aspect that has attracted some attention (IEEE LTSC, 2001; Sampson, Karagiannidis & Cardinali, 2002).

A lesson that we have learned is that courseware systems development is not an isolated activitiy – there is usually a strategic dimension. Embedded into an educational, technical, and organisational context, courseware systems nowadays need to be *engineered* to meet the requirements of their environment. Engineering means to address the needs of all stakeholders, it means to consider a system's environment and the interactions and dependencies between the system and its environment. The *educational, organisational,* and *technical environments* and the *variety of evolving roles*, including learner, instructors, instructional designer, software developer, and administrators, form this specific context. Our conclusion is that only a methodology for courseware development can help us to address in particular context-specific problems that we and others have encountered in the past and that prepares us for the problems that we anticipate for the future.

References

ADLNet. (2004). SCORM Standards. http://www.adlnet.org.

Alspaugh, T.A., Anton, A.I., Barnes, T., & Mott, B.W. (2001). An Integrated Scenario Management Strategy. In *Proc. IEEE International Symposium on Requirements Engineering*, 142-149. IEEE Press.

Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice*. Addison-Wesley.

Beynon-Davies, P., & Holmes, S. (2002). Design breakdown, scenarios, and rapid application development. *Information and Software Technology*, 44:579–592.

Bødker, S. (2000). Scenarios in user-centred design – setting the stage for reflection and action. *Interacting with Computers* 13(1): 61-75.

Boyle, T. (2003) Designing principles for authoring dynamic, reusable learning objects. *Austral. Jrnl of Educ Tech* 19(1): 46-58.

IEEE Learning Technology Standards Committee LTSC (2001). *IEEE P1484.1/D8. Draft Standard for Learning Technology – Learning Technology Systems Architecture (LTSA), 04/06/2001.* IEEE Computer Society.

IEEE Learning Technology Standards Committee LTSC (2002). *IEEE P1484.12/D4.0 Draft Standard for Learning Object Metadata (LOM).* IEEE Computer Society.

Pahl, C. (2002). An Evaluation of Scaffolding for Virtual Interactive Tutorials. *Proc. 7th E-Learn 2002 Conference*, AACE.

Pahl, C. (2003). Managing evolution and change in web-based teaching and learning environments. *Computers & Education* 40(1):99-114.

Pantano Rokou, F., Rokou, E., & Rokos, Y. (2004). Modeling Web-based Educational Systems: Process Design Teaching Model. *Educational Technology & Society* 7(10):42-50.

Sampson, D., Karagiannidis, C., & Cardinali, F. (2002). An Architecture for Web-based e-Learning Promoting Re-usable Adaptive Educational e-Content. *Educational Technology & Society* 5(2).

Xu, L., Pahl, C., & Donnellan, D. (2003). An evaluation Technique for Content Interaction in Web-based Teaching and Learning Environments. In *Proceedings International Conference on Advanced Learning Technologies ICALT 2003*. IEEE Press.

Virvou, M., & Tsiriga, V. (2001). An object-oriented software life cycle of an intelligent tutoring system. *Journal of Computer Assisted Learning*, 17, 200-205.