Reuse- and Aspect-Oriented Courseware Development

Khaldoun Ateyeh and Peter C. Lockemann
Fakultät für Informatik, Universität Karlsruhe, Postfach 6980, 76128 Karlsruhe, Germany
ateyeh@ipd.uka.de
lockeman@ipd.uka.de

ABSTRACT

No longer can courseware providers deal with one homogeneous target group, one learning form and possibly one pedagogical approach. Instead they must develop a broad range of courseware, each serving its specific target group, each adjusted to a specific learning and teaching form, each appealing to its own learning and teaching scenario, and each incorporating its own pedagogical approach, and to do all this in a cost-effective and timely fashion. The thesis of this paper is that only an approach that is much more dictated by software engineering principles than what has been usual so far will meet these needs. Because of the economical constraints, the overriding engineering principle should be component reuse, and if several distinctive concerns become interwoven – above all content, didactics and technology – component reuse should be augmented by aspect-oriented programming. The paper develops and details a novel courseware engineering process that combines software reuse, component technology and aspect-oriented programming.

Keywords

Courseware engineering, Courseware reuse, Aspect-oriented development, Educational system development

Introduction

The goal of modern instructional design and learning theory are curricula that no longer follow a standardized pattern but are customized to the students' prior knowledge and learner type. But the goal also raises new challenges to courseware providers. No longer can they deal with one homogenous target group, one learning form and possibly one pedagogical approach. Instead they must develop a broad range of courseware, each serving its specific target group, each adjusted to a specific learning and teaching form, each appealing to its own learning and teaching scenario, and each incorporating its own pedagogical approach, and they must do all this in a cost-effective and timely fashion.

This makes courseware development essentially an engineering challenge. It is a challenge, though, that has long been known to software engineering. There, component reuse is considered the key technique. Our vision, then, is to apply this technique to the creation of courseware that is highly adaptable and modular, and thus reusable in many contexts and for many needs. It should be possible to easily adapt the same courseware so that it can be used by students at universities and by employees in companies, by the student who prefers to learn online as well as by the one who prefers the traditional way of learning by printing out a script, by an instructor in the lecture hall or by one in a virtual class room. Stated differently, the way courseware components are to be reused depends on further factors such as pedagogical, psychological, and ergonomic aspects. Observing such factors during the development of component-base software has been known in software engineering as aspect-oriented programming.

This paper demonstrates that despite the differences the engineering techniques of software reuse and aspect-oriented programming can successfully and profitably be applied to courseware development, although they need to be specialized for the purpose. The paper is organized as follows. After examining the facets of reuse and aspect-orientation and the relevant state-of-the-art we develop the main ideas of a reuse-driven courseware development process. We introduce a model development process that allows us to separately pursue the different educational concerns in courseware. In particular we treat domain engineering and its specialization to content and didactics, and discuss the weaving of the concerns (or aspects) into a complete courseware. The final chapters cover the implementation and the conclusions.

Software engineering for courseware

The thesis of this paper is that courseware development can profit from certain principles of software engineering, specifically software reuse and aspect-oriented programming. We now take a closer look at the implications of these principles.

Software reuse has two dimensions, building a stock of software components and combining components into a system with the desired properties. The first dimension is referred to as *engineering for reuse* and the second as *engineering with reuse*. Correspondingly in courseware development, engineering for reuse would refer to the design and development of learning assets (any reasonably delimited material which can contribute to a learning object) so that they can be reused later on when a specific teaching or learning environment is to be served, and engineering with reuse to synthesizing a particular courseware.

Now, suppose a repository of learning assets that has been built up over time. Imagine a teacher or learner who needs instructional material on a certain subject. In all likelihood he/she will inspect the repository for assets that cover the desired *content*. Suppose a suitable asset has been found. Then the particular circumstances of the teacher or student come into play. For one, these determine the *didactics* to be pursued, that is, the content should be organized to account for the expected background and experience, the desired learning form and scenario, and the pedagogical approach. For another, the circumstances dictate how the material is to be accessed and – above all – how it is to be *technically presented*, e.g., visually by static or animated graphics, video sequences, audio, or several in combination, and what technical arrangement to choose. Content, didactics and technology constitute what we call *aspects*. Courseware development must observe these aspects. It is the combination of component reuse and aspect observation that seems to pose major particular challenges.

Existing contributions to the problem

Software engineering contributions

The literature distinguishes between two main categories of software reuse approaches, component-based reuse and generation-based reuse (Biggerstaff & Perlis, 1989; Henninger, 1997; Szyperski, 2003a). A component-based development process describes all activities in the context of a complete software life cycle on the basis of components. A software component has contractually specified interfaces and explicit context dependencies, can be deployed independently, and is subject to composition by third parties (Szyperski, 2003b). This matches our goal of deploying entire learning objects and configuring them into larger courses, following, e.g., the SEI reference model (SEI, 2004).

Component-based reuse, or more precisely systematic, reuse-based development of system families in a domain rather than one-of-a-kind systems is the objective of *Domain Engineering* (DE). As defined in AOSD (2003), domain engineering is the activity of collecting, organizing and storing past experience in building systems or parts of systems in a particular domain in the form of reusable assets, as well as providing an adequate means for reusing these assets when building new systems. What a domain is in a specific situation is left to the consensus of the so-called stakeholders in that domain. The practical goal of the domain engineering process is to build a reference architecture that can be easily reused across members of a system family or families in that domain. Domain engineering appears particularly attractive to us, since its objective seems to concur with the engineering for reuse phase.

Generation-based reuse takes a higher-level specification of a software artifact and produces its implementation via a generator. The users of a generator see a system that allows them to go from a specification to a software component without having to understand the internal details of the generator (Bell et al., 1994). A specific representative of generation-based reuse, *Aspect-Oriented Programming* (AOP), seems particularly suited to our problem of separating and integrating aspects. AOP deals with separation of concerns at the implementation level and tries to provide linguistic mechanisms to factor out different aspects of a program, which can be defined, understood, and evolved separately (Czarnecki & Eisenecker, 2000). The goal is to provide methods and techniques for decomposing problems into a number of functional components as well as a number of aspects that cut across functional components, and then compose these components and aspects to obtain a final system implementation (IBM, 2004). Unfortunately, the definition sounds very abstract, and even the three steps of aspectual decomposition, concern implementation and aspectual re-composition by weaving remain abstract principles. Not surprisingly, then, AOP is more an open research agenda than an existing technology that one can readily use. First approaches are subject-oriented programming (SOP) (IBM, 2004), Adaptive Programming (AP) (Lieberherr, 1996), and Composition Filters (CF) (Aksit, 1989), all of them tailored to object-oriented programming. So while the concepts sound attractive to our problem, we would need to invest into further research.

Systematic courseware development

Traditional methods of courseware development have a tendency towards courseware for specific learning situations, learner characteristics, learning objectives, or learning/teaching strategies. Consequently, there has been scant need for systems that are adaptable to varying needs. Rather the result is a monolithic structure that mixes different aspects such as content, didactic, and technical aspects in inseparable ways, and where one of these aspects dominates the others.

Take the didactic-oriented model that concentrates on the didactic aspect of courseware development. Most prominent is the instructional design (ID) model that tends towards the design and development of so-called tutorial and drill-and-practice systems (see, e.g., Tennyson & Rasch, 1995; Merrill et al., 1990). While considered outdated by today‘s learning/teaching methods objectives, the model still is interesting because of its strength in its methods for analyzing learning situations, characteristics of the learners, learning objectives, and for designing suitable learning/teaching strategies. Or as another example, take technology-oriented models that focus on the technical and engineering aspects of courseware under the assumption that the use of multimedia and/or hypermedia as well as technical tools will significantly enhance the learning/teaching process. Typical examples can be found in (Boles & Schlattmann, 1998; Garzooto et al.,1991; Isakowitz et al., 1995) or as commercial products (Macromedia, 2004a; Macromedia, 2004b; ToolBook, 2004). What these models have to offer to other approaches, though, are the sophisticated authoring environments and a structured development process divided into phases of information objects production, authoring, and generation. Courseware engineering models seek to combine the strengths of the two other models. They consider courseware development as a mixture of software development along the line of a process of developing business software, and instructional design as the process of developing instructional or didactic models. However, the main focus of these models is to provide a systematic development process that allows one to manage the growing complexity of courseware development. Little or no consideration is given to courseware reuse. Examples are the Essener-Learn-Model (Pawlowski, 2001) and the IntView Lifecycle Model (Grützner et al., 2002) as well some recent research results (Blumstengel, 1998; Klein, 2002).

Even a focused, monolithic system can find many users provided it is accessible from outside places. This has been recognized over the past years by the eLearning community which has come up with various standards in order to ensure reuse and interoperability. Relevant standards are the Learning Object Metadata (LOM) standard (IEEE, 2003), the IMS Content Packaging (IMSCP) standard (IMS, 2004), the IMS Learning Design standard (IMSLD, 2004), the Learning Technology Systems Architecture (LTSA) standard (IEEE, 2001), and the Sharable Content Object Reference Model (SCORM) (ADL, 2004). The standards cover nearly all aspects of learning such as didactic, content, learner profile, learning management system, and their use seems essential for the support of reusability at an inter-community/inter-organization level and of interoperability among courseware/learning management systems.

Clearly, any effort towards more flexible courseware development should take cognizance of these standardization efforts even though their large number certainly is confusing. In addition, they make assumptions on the development process that do not seem entirely realistic. One assumption is that a component can be reused in new contexts just as it is. This seems unrealistic for the content and didactic aspects: In our experience these seem highly interdependent, so that one cannot simply transfer a component into a different didactic environment without adapting its content. Under IMSLD, even the instructional or didactic models are designed with a specific learning context in mind so all one can do is reuse the corresponding components if their context happens match the selected or designed instructional strategy.

There has been some work on generating courseware from components, given they fit the requirements. Merrill introduces the concept of instructional transaction shell to organize existing so-called instructional transactions and knowledge objects into new courseware (Merrill et al., 1990). On the other hand, little is being said about the process of the development of such reusable elements, rather the focus is on the process of automatic generation of instructions from existing reusable instructional transactions and knowledge objects.

In conclusion, we face a number of challenges. It should be possible to design courseware in small units, with a clear separation of the various aspects, in our case predominantly contents and didactics. Their interdependence should be taken into account when courseware is produced for a specific learning/teaching context. Consequently, there should be a clear separation of the modeling phase – corresponding to engineering for reuse – and the production phase – corresponding to engineering with reuse. The production phase should largely be automated, somewhat in the sense of model-driven programming.

Courseware development process model

The challenges as discussed in the previous section seem to suggest a crude process model consisting of two major processes, engineering for reuse and engineering with reuse. Figure 1 gives an overview. The two processes are referred to as *domain engineering* and *course engineering*. Courseware engineering must start with domain engineering because it is this process that lays the foundation for course families and, hence, for a variety of courseware even for the same topic, where the differences reflect the situation to be served. Only when a family has been established – at least in part – can individual family members be developed. In practice, though, the processes are not strictly separated but iteratively performed. For example, if we assume a repository to hold all developed assets the repository may not entirely satisfy all current needs so that one may decide to ignore an asset and build a new, suitable one from scratch. The result will be stored in the repository and thus become part of some other development.

Both processes are themselves iterative processes. The domain engineering process is relatively independent of the course engineering process whereas the course engineering process strongly depends on the results of the domain engineering process. Therefore, there is only limited potential for conducting the two in parallel. On the other hand, new requirements that have a general character and have not yet been considered in the domain engineering process but are first discovered in the course engineering process are fed back to the domain engineering process.

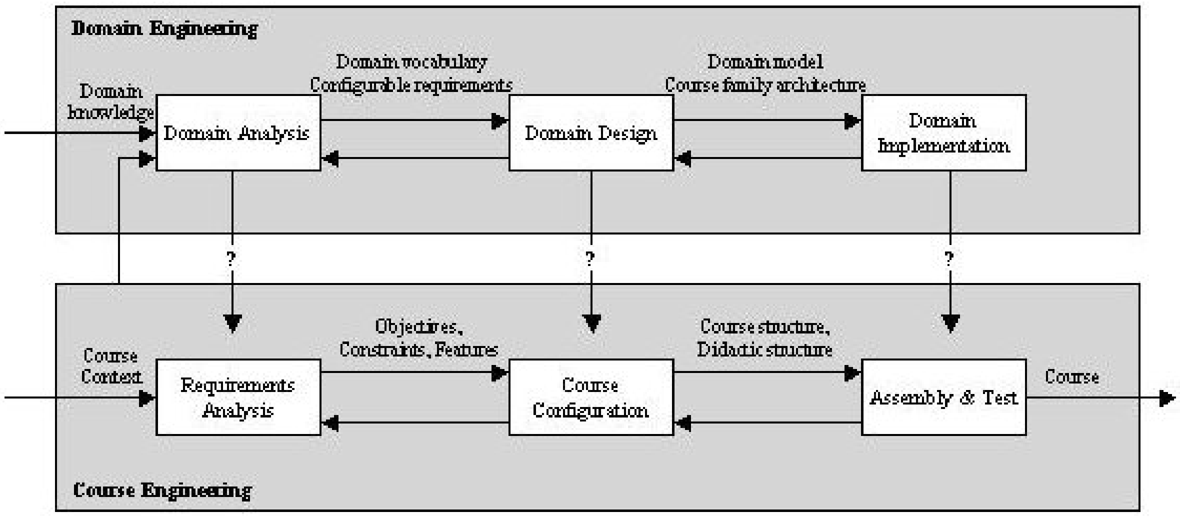


Figure 1. An overview of the development process

Both processes are further divided into phases. The division according to Figure 1 is only effective if, beyond associating clear milestones with each phase and allowing for collective development by the various experts, one can clearly identify which phases from domain engineering contribute which results to which phases of the course engineering process. The answer to this problem is the subject of the remaining chapters.

How do the software engineering techniques we plan to employ contribute to the courseware development process?

- Domain engineering is geared towards the development of families within a given knowledge domain.
- Aspect-oriented programming allows one to concentrate on different aspects one at a time (a principle often also referred to as separation of concerns).
- Component technology allows one to assemble the learning assets into larger but still self-contained courses.

Domain engineering is the general principle that underlies our process model. Aspect-oriented programming gives substance to the principle and will affect all three phases of the domain engineering process. We consider three aspects:

- The contents of the course (content aspect): The instructor needs to decide on a syllabus of the course, choosing which (sub-)topics to include, how much emphasis to put on each, and deciding on an order of presentation.
- The didactic strategy and methods to use (didactic aspect): The instructor needs to decide which didactic strategy is suited best to reach the objectives of the course.

- ➢ The learning/teaching context or constraints under which the course will be taught (technical aspect): These include the number of participants, the amount of time available, the room and the technical equipment that will be used, whether the course will be face-to-face or distance learning, whether teaching will be synchronous or asynchronous and so on. We will not discuss the technical aspect in detail but include it in the didactic aspect whenever appropriate.

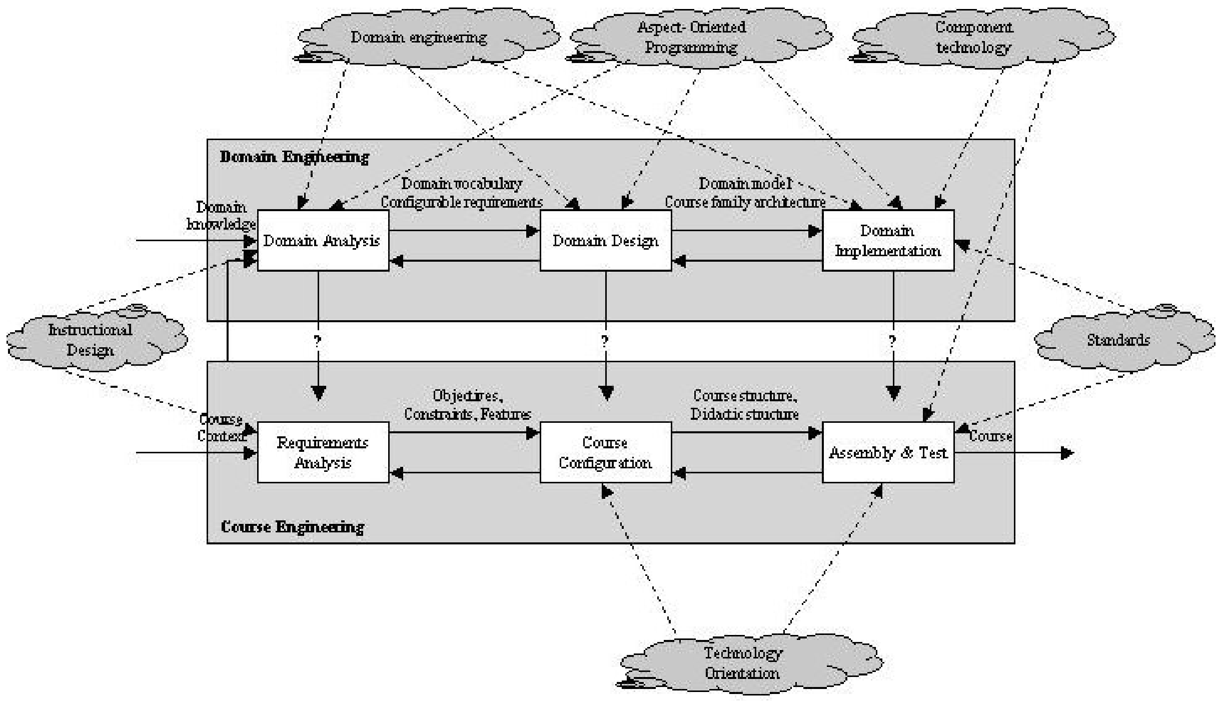


Figure 2. Influence of existing techniques on the development process

Finally, component technology is the basis for constructing the learning objects for a specific course from the assets and should follow one or more of the established techniques for constructing components.

Figure 2 summarizes how the three techniques impact the domain engineering process and where, according to the earlier discussion on systematic courseware development, instructional design and standards exert an influence.

The domain engineering process

The gross structure of domain engineering consists of the three phases of domain analysis, domain design, and domain implementation. All three are mainly dominated by considerations of the domain knowledge – the contents – and of three independent aspects – content, didactics, and technology. Below we study the effects mainly of the first two factors on each phase. A course on database systems will illustrate the development process.

Domain analysis

Content analysis

During *content analysis* the scope of the knowledge domain is selected and defined. It is the responsibility of a larger community to agree on the general topics that should be covered by the domain model as well as those to be left out. Figure 3 gives some examples. The main contributions come from material used by the community members in conducting their own courses, from reference textbooks, from knowledge domain experts in the community, and perhaps from available knowledge domain ontologies. As part of the discussion one should refine the general concepts into more specific ones, assign terms to them and collect these terms in a vocabulary.

Didactics analysis

This phase considers both, content didactics and process didactics. The latter covers general properties such as teaching/learning goals and contexts, potential audiences. The former focuses on the definition and selection of the various *didactic principles* through which the learning materials will be submitted to the students. Figure 3 shows some examples.

Obviously, didactic principles are generic, i.e., independent of a specific knowledge domain. Consequently, they can be specified just once and early on, and can then be applied to a broad range of knowledge domains.

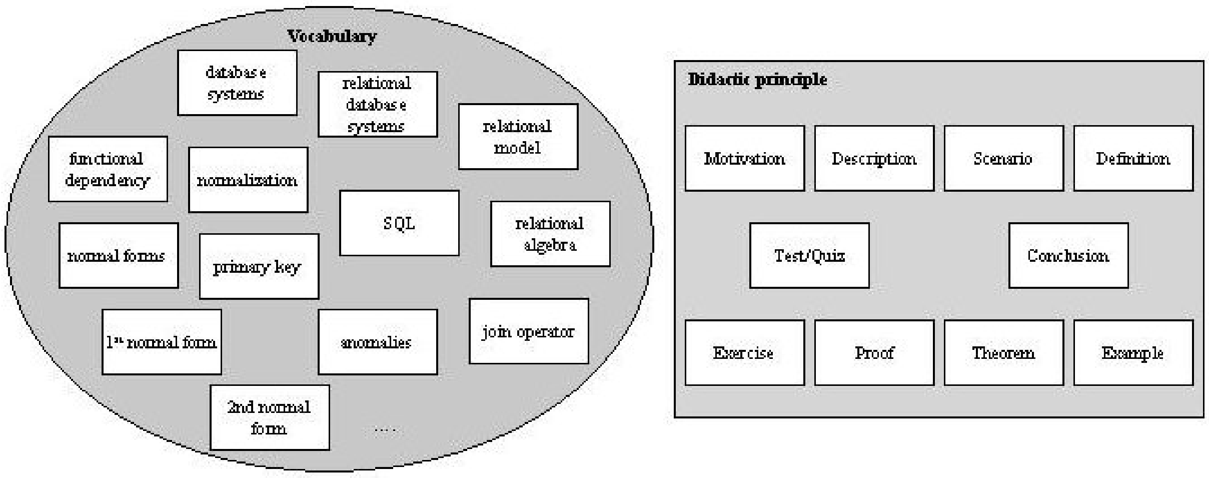


Figure 3. Sample result of the domain analysis

Domain design

The purpose of the domain design phase is to develop a more formal courseware model from which one can systematically derive an implementation.

Content design

Content design defines a shared content model for the content concern. The content model aims at the creation of a shared understanding for the knowledge domain that is of interest to the members of the community independent of its use in a specific learning context (Ateyeh et al., 2003). Basically one determines the conceptual entities that form a common basis for the reuse and exchange of courseware content, where each unit can be (re)used by each member of the community.

Ontologies appear well-suited for the purpose. An ontology is an explicit specification of a conceptualization, where a conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. The purpose of an ontology is the shared understanding of some domain of interest (Uschold & Gruninger, 1996). It usually takes the form of a set of concepts (e.g. entities, attributes, processes, etc.), their definitions, and their relationships.

The basis of a knowledge domain ontology are the concepts identified during analysis. These are organized into a relational structure, with the relationships dictated by the purpose of courseware. Two types of relationships seem sufficient:

- isSubTopicOf: Two terms A and B have relationship B *isSubTopicOf* A if A is more generic than B. E.g., "relational algebra" is more generic than "join operator": join operator *isSubTopicOf* relational algebra.
- isPrerequisiteFor: Two terms A and B have relationship A *isPrerequisiteFor* B when the knowledge of A is needed to understand B. In our example, knowledge about functional dependencies is needed before tackling normalization: functional dependency *isPrerequisiteFor* normalization.

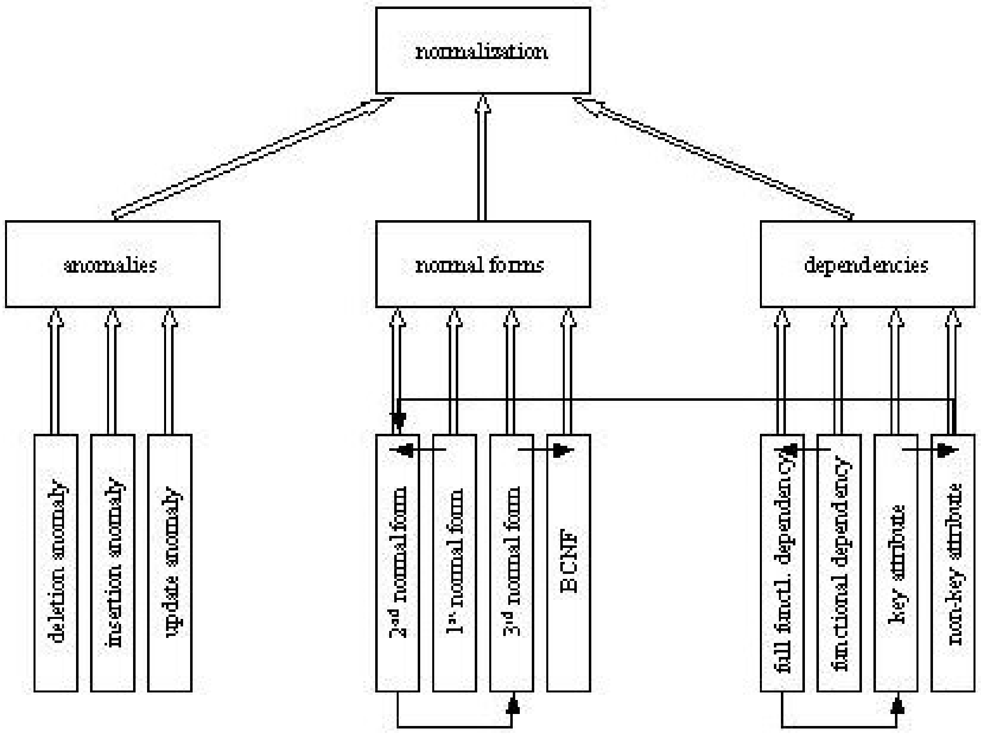


Figure 4. Sample domain knowledge ontology (White arrow: isSubTopicOf, Black arrows: isPrerequisiteFor, both relationships are transitive)

Designing an ontology is by no means trivial, nor are there commonly accepted standard methods for building ontologies. We refer the reader to the work of Holsapple and Joshi (2002) on collaborative ontology design.

Figure 4 shows a very small part of an ontology for the domain "data base systems" that captures the topic of normalization. The resulting ontology can then be implemented using one of the many ontology representation languages such as RDF/RDFS, OWL, DAML, KIF, etc. The different representation languages provide different levels of formality to capture different kinds of ontology characteristics and semantics. We found RDF to be sufficient four our purpose.

We note that content analysis and content design have to be done just once for each knowledge domain. Ideally, one could then defer the combination of ontology and didactic principles to the domain implementation phase.

Content didactics design

As pointed out earlier, content and didactics are interdependent so that one cannot simply transfer a component into a different didactic environment without adapting its content. Indeed, we found it extremely difficult to delay their combination past content didactics design. Instead we merge the ontology with the didactic principles, i.e., the intended didactic usage of the content, into what we call *learning atoms*. Figure 5 gives a few examples. Figure 5 also demonstrates that atom types may belong to terms on any level of the ontology.

Learning atoms that belong to the same term in the ontology are combined into a *learning module*. Hence, for each term in the ontology there exists exactly one corresponding learning module with all learning atoms related to that term. As we move upwards to higher levels of the ontology, more complex modules are constructed that include modules from the lower levels. Consequently, the notion of learning module is recursively defined. Figure 5 sketches an example.

Since there is a learning module for each term in the ontology, modules are already implicitly defined during the creation of the ontology. By instantiating an atom from a certain atom type it automatically becomes part of the appropriate module (and implicitly also of the higher-up modules through the isSubtopicOf relationship).

Process didactics design

We focus on two aspects of the so-called learning process didactics:

- The Learning/teaching strategy: Structures the learning/teaching process into one or more phases. Each phase meets a specific didactic goal. For every learning/teaching phase a suitable didactic method is used.
- The Didactic method: Is a system of didactic rules that instructs or guides students or teachers while learning or teaching. A didactic method achieves certain learning or teaching goals under given circumstances.

As an example for a learning/teaching strategy take the problem-based learning/teaching strategy of Merrill rooted in the constructivism learning theory (Merrill, 2000). A second example and the one we use to illustrate our approach is the three-step learning/teaching strategy of Figure 6 with three main phases:

- Introduction phase: It defines the starting point and the goals of the learning process.
- Working phase: This phase consists of further sub-phases (didactic functions): setup, workthrough, apply, transfer, assess and integrate, and should be supported by a suitably stimulating environment.
- Completion phase: The goals of the learning process must be secured.

Independent of the didactic functions one can define didactic methods. These are shown in Figure 6 below the solid line to indicate that, at least in principle, each may be associated with each didactic function.

In the literature, learning/teaching strategies and didactic methods are mostly (if at all) expressed in natural language. Clearly this is not suitable in a cooperative environment that tries to increase courseware reuse and that should be based on automatic processing of courseware entities. What is needed is some kind of formalism that allows to capture the didactic concern and to easily adapt them to a given learning/teaching context. Figure 7 graphically illustrates our own didactic metamodel. The figure also indicates (shaded box) where the didactic methods are connected to the didactic functions. Further, the figure demonstrates how to integrate the technical aspect with the didactic aspect. The implementation has been done in XML.

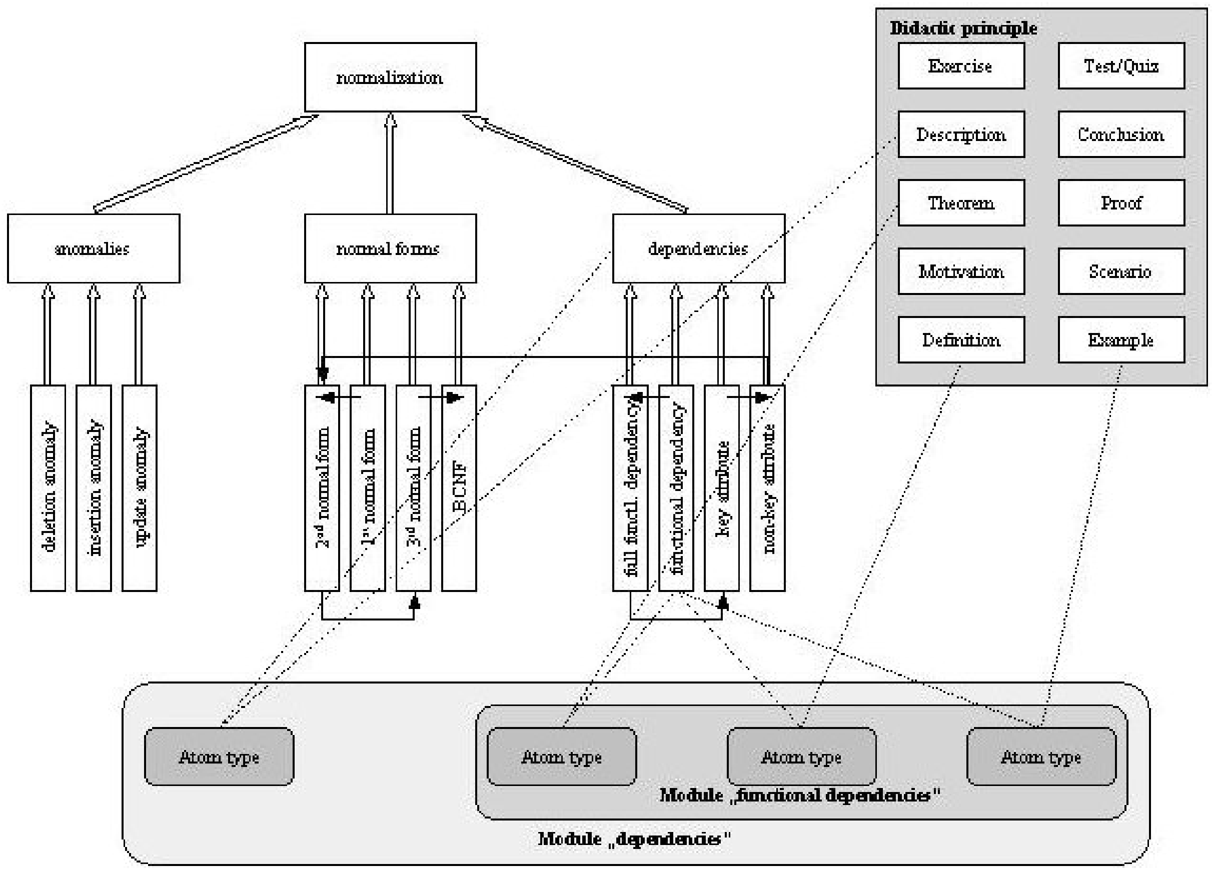


Figure 5. Sample learning modules

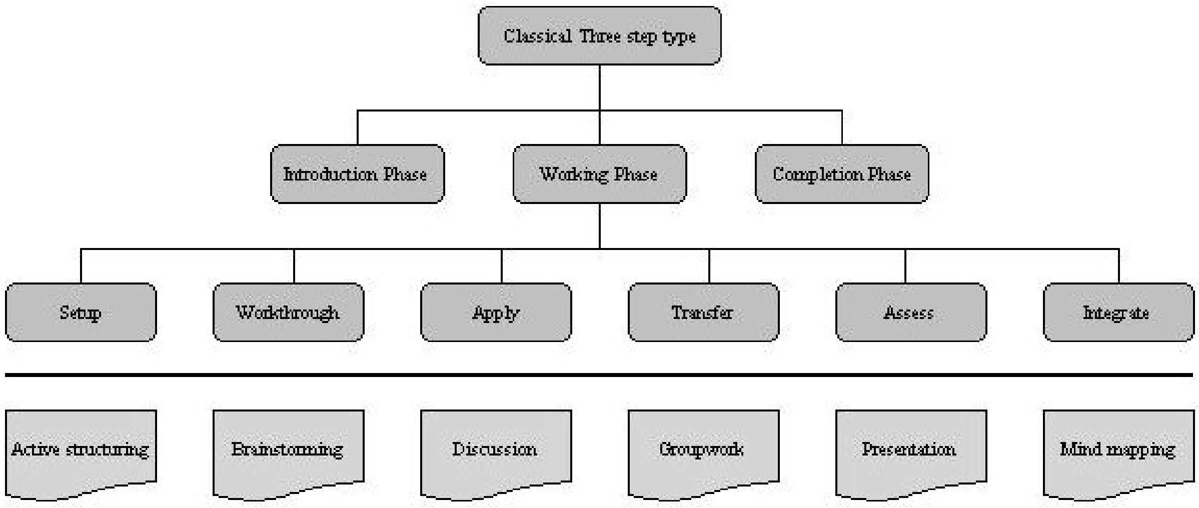


Figure 6. Classical three-step learning/teaching strategy and associated didactic methods

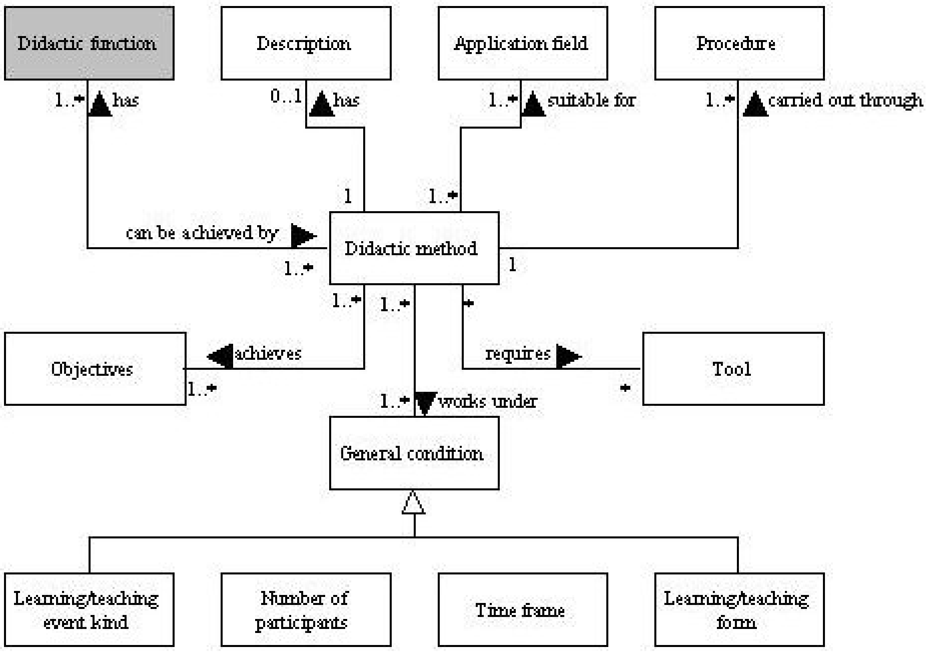


Figure 7. The didactic method metamodel

Domain implementation

In this phase concrete reusable courseware entities (learning assets) for the models defined in the design phases (content entities, didactic entities, software tools) are implemented and then kept in a repository for subsequent use in the course engineering process. They serve as a starting kit for the courseware development in the course engineering process.

Content implementation

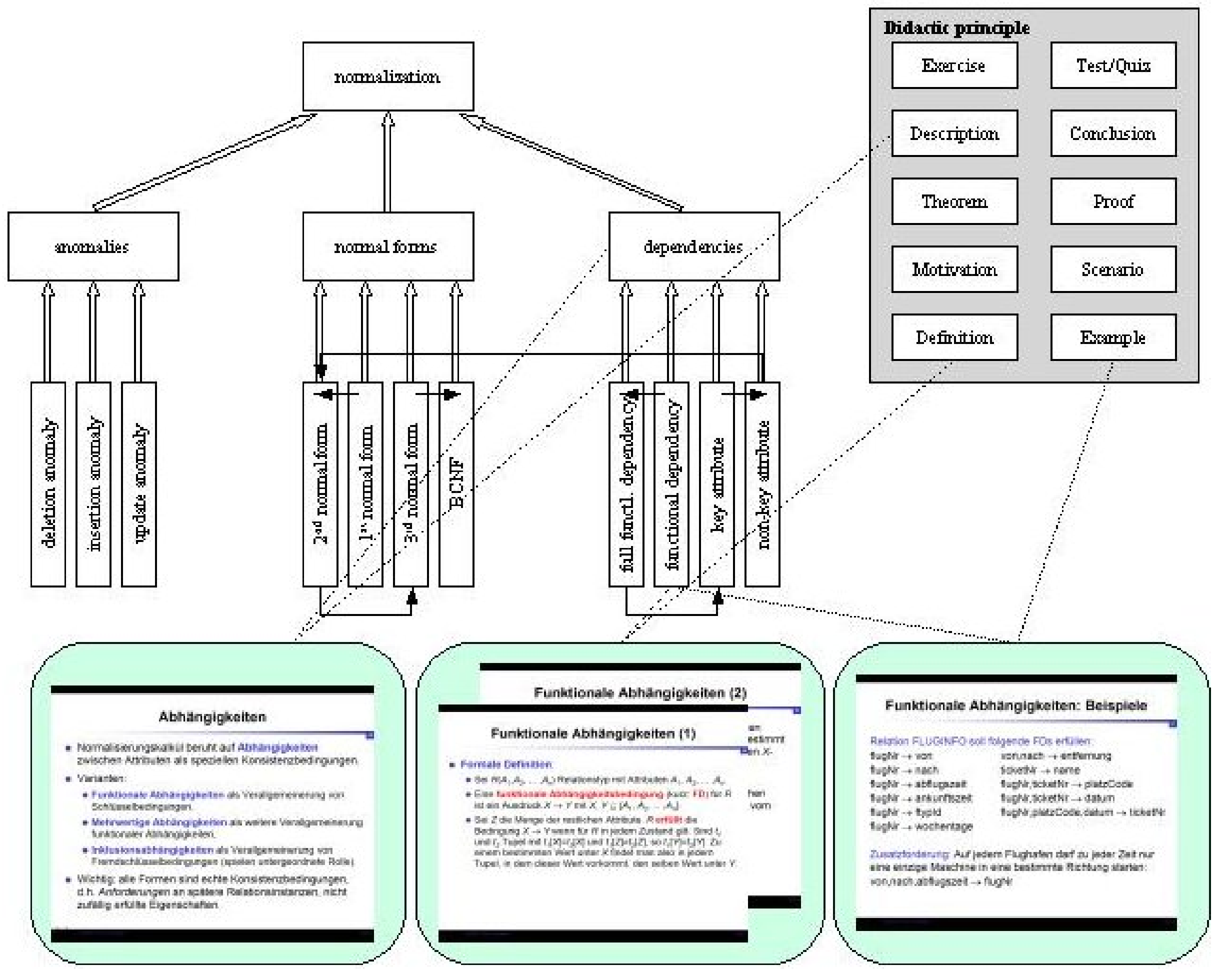
Strictly speaking, what we defined in the design phase are learning atom types and module types. Atom types are instantiated to one or more learning atoms: The content is developed in detail and perhaps with different foci depending on the background of the learner and the intentions of the teacher, representations are chosen such as powerpoint slides or pdf files, and both are combined into an executable unit in a form that can be reused in different environments and situations (for an example, see Figure 8). The atoms are stored in a *content repository*.

The repository is organized into containers. A container maintains the contents of a learning module. Consequently, there exists a container for each term in the ontology. At a minimum, for a lowest-level term a container consists solely of learning atoms. In general, a container may hold atoms as well as "smaller" containers for the less generic modules. Taking the example of Figure 5, there are containers for functional dependency (with atoms only), dependencies (it would include the functional dependency container), normalization, etc.

Didactics implementation

In a sense, domain design "atomizes" the learning material into a combination of local contents and local didactics. However, learners are to be offered self-contained courses. A course is subject to a *content didactic strategy*. Such a strategy starts from a didactic function, determines the associated didactic method and defines in the form of a procedure (Figure 7) the didactic principles through which to reach the students, and is represented by a so-called *didactic template*. The central idea of our approach is to include in the repository a number of such templates as assort of preplanning the potentially desired course structures. Figure 9 gives an example for a didactic template that could be used for a presentation during the setup phase. Suppose we apply it to an SQL course module. The template states that the course should begin with an overview followed by a motivation and an activation atom. Subsequently, for each sub-topic of the course an explanation or a definition of the topic is needed followed by an example. Finally the course should finish with a concluding example.

Since our goal is to automatically generate a course from the content repository and a given template, templates should be formulated in a machine-processable script language. For the purpose, we developed DidaScript, a typical script language with the usual constructs for variable declarations, assignment statements, conditional and loop expressions. DidaScript includes built-in functions for the navigation in a course structure of learning modules, for ordering all or selected atoms of a specified module, and for hiding atoms in a module (Ateyeh, 2004). The resulting course structure is represented via XML.



Abhängigkeiten

- Normalisierungskalkül beruht auf Abhängigkeiten zwischen Attributen als speziellen Konsistenzbedingungen.
- Varianten:
  - Funktionale Abhängigkeiten als Verallgemeinerung von Schlüsselbedingungen.
  - Mehrwertige Abhängigkeiten als weitere Verallgemeinerung funktionaler Abhängigkeiten.
  - Inklusionsabhängigkeiten als Verallgemeinerung von Fremdschlüsselbedingungen (spielen untergeordnete Rolle).
- Wichtig: alle Formen sind echte Konsistenzbedingungen, d.h. Anforderungen an spätere Relationsinstanzen, nicht zufällig erfüllte Eigenschaften.

Funktionale Abhängigkeiten (2)

en
estimmt
en X-
hen
vom

Funktionale Abhängigkeiten (1)

- Formale Definition:
  - Sei $R(A_1, A_2, \ldots, A_n)$ Relationstyp mit Attributen $A_1, A_2, \ldots, A_n$
  - Eine funktionale Abhängigkeitsbedingung (kurz: FD) für R ist ein Ausdruck $X \rightarrow Y$ mit $X, Y \subseteq \{A_1, A_2, \ldots, A_n\}$.
  - Sei Z die Menge der restlichen Attribute. R erfüllt die Bedingung $X \rightarrow Y$ wenn für R in jedem Zustand gilt: Sind t_1 und t_2 Tupel mit $t_1[X]=t_2[X]$ und $t_1[Z]\neq t_2[Z]$, so $t_1[Y]=t_2[Y]$. Zu einem bestimmten Wert unter X findet man also in jedem Tupel, in dem dieser Wert vorkommt, den selben Wert unter Y.

Funktionale Abhängigkeiten: Beispiele

Relation FLUGINFO soll folgende FDs erfüllen:

flugNr → von	von,nach → entfernung
flugNr → nach	ticketNr → name
flugNr → abflugszeit	flugNr,ticketNr → platzCode
flugNr → ankunftszeit	flugNr,ticketNr → datum
flugNr → flypId	flugNr,platzCode,datum → ticketNr
flugNr → wochentage	

Zusatzforderung: Auf jedem Flughafen darf zu jeder Zeit nur eine einzige Maschine in eine bestimmte Richtung starten:
von,nach,abflugszeit → flugNr

Figure 8. Sample learning atoms

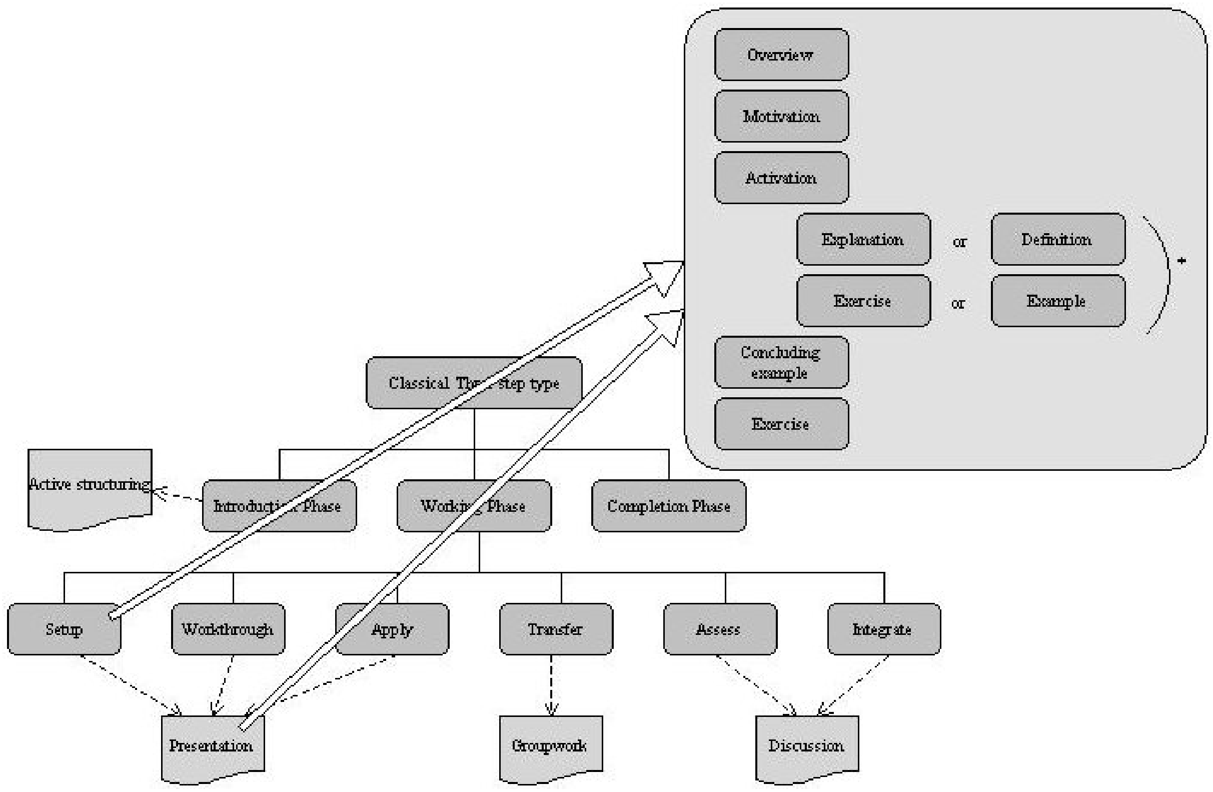
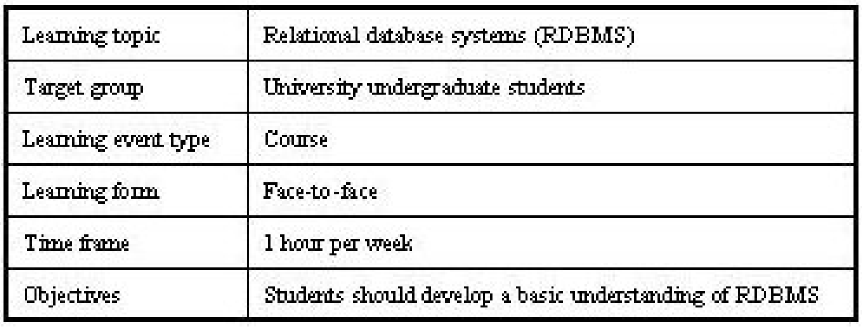


Figure 9. The classical three-step learning/teaching strategy augmented by didactic methods and the association of a content didactic template



Learning topic	Relational database systems (RDBMS)
Target group	University undergraduate students
Learning event type	Course
Learning form	Face-to-face
Time frame	1 hour per week
Objectives	Students should develop a basic understanding of RDBMS

Figure 10. A sample course specification

The course engineering process

Domain engineering is engineering *for* reuse and results in a host of learning assets and organizes them in an organizational structure. Course engineering is engineering *with* reuse and employs the results of the domain engineering process to construct from them, with moderate effort, context-specific courses in the form of learning objects (We use the term "course" as a general notion that refers to any learning/teaching event such as course, seminar, lesson, etc.).

We pursue two goals. One is to fuse the outputs of content engineering and didactics engineering (or in AOP jargon, to weave them). The second goal is to do the fusion as much as possible in an automated fashion, by drawing on the repository.

Requirements analysis

The goal of this phase is to analyze and define the (general) requirements for the course, such as the learning/teaching topics, target group, objectives, learning form, time frame etc. In this phase both results from the analysis and the design phase of the domain engineering process are used. The requirements are expressed through so-called general specifications for the learning/teaching event (see Figure 10 for an example).

Course configuration

Didactics configuration

Based on our initial experience we recommend to start the course configuration with didactics. As a first step the requirements are re-formulated as a learning strategy. Consider nested queries as a subtopic of an SQL course. Using the classical three-step learning/teaching strategy we proceed as follows: In the introduction phase, existing knowledge about SQL should be recapitulated, and it should be motivated why nested queries are needed. In the working phase, the syntax of nested queries should be introduced (set-up), some examples should be shown (work-through and apply), given natural language requests for specific information nested queries should be formulated (transfer), nested queries should be compared to alternative ways to obtain the same information (assess) and nested queries should be used together with other SQL constructs (integrate).

In the next step the learning strategy must be translated into an assignment of didactic methods. To return to our example of nested queries, the introduction phase could require students to try and formulate queries with the SQL constructs they already know, revealing the need for nested queries (active structuring). Then, a presentation of the syntax of nested queries followed by the online presentation of some examples could cover the set up and apply phases, the students could then individually formulate nested queries to gain information, a collaborative discussion comparing nested queries to alternative formulations would be an appropriate method for the assessing phase. A computer session, requiring the usage of nested queries together with other SQL constructs could serve both to integrate and to secure the knowledge gained (Figure 9 is a good illustration).

Given these assignments, we choose an appropriate – hopefully predefined – didactic template from the repository. The template in turn defines the didactic principles to be applied to the material. Together with the content structure we now know which learning atoms and modules to select from the repository.

Content configuration

During content configuration we have to decide on the precise content (learning modules). The process of didactics configuration gives numerous suggestions on how to organize the content. Therefore, didactic and content configuration run somewhat in parallel. Remember that a learning module contains all types of material from all participants that can be associated with the corresponding term in the ontology. Consequently, the course that evolves from choosing the topics (modules) from the ontology reflects the structure predefined by the *isSubtopicOf* relationship of the domain-specific ontology. The instructor can now adapt this structure to his requirements. In particular, she/he selects from the predefined structure those sub-topics (modules) she/he intends to address in her/his course, and defines structural relationships between the selected topics which define the navigation structure that has to be used by the students and/or the instructor. The development platform may support her/him by warning her/him to avoid a structure that would violate the relation *isPrerequisiteFor* defined in the domain-specific ontology. To continue our earlier example, we would obtain for the part of the course that deals with normalization the sequential structure of Figure 11. Comparing it with Figure 4 we note that all direct sub-modules of *normalization* were selected, whereas for module *normal forms* the sub-module *BCNF* has been deleted.

Weaving a course

Given the template and the course structure, we can mechanically draw the necessary learning atoms and modules from the repository and weave them into a course flow. Automation is in the form of a generator. Concrete learning atoms can be drawn from the repository by the generator merging, in a stepwise fashion, the learning atom type from the content structure and the didactic principle from the didactic template. Figure 12 illustrates the principle for our normalization example. The course generator selects from the root learning module only those learning atoms that follow either the didactic principle of overview or motivation. These are then ordered so that the overview atoms precede the motivation atoms. Further, the strategy states that for each sub-module of the root module an explanation or a definition of the topic should be given followed by an exercise or an example. Finally the course generator visits the root learning module again to select a concluding example followed by an exercise and the solution for that exercise.

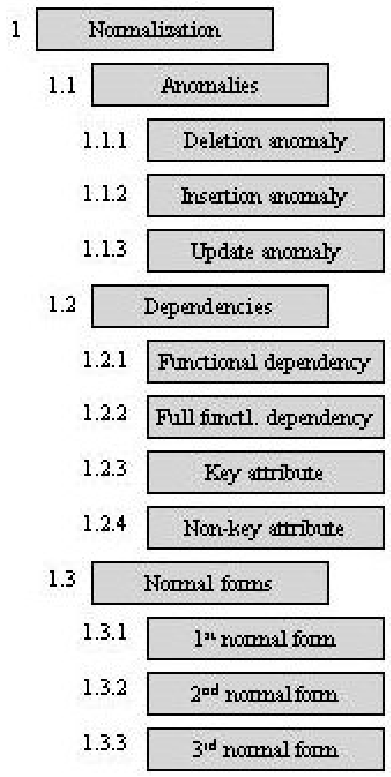


Figure 11. Sample course structure

Assembly, test and integration

The course modules resulting from the configuration phase are normally half-finished modules. E.g., for some learning topics appropriate learning atoms may not be found in the repository. In this case, an appropriate learning atom must be developed and added to the repository for future use. In this case, the instructor switches back into his role as an author, develops the appropriate material and adds it to the repository. Furthermore, in many cases it is necessary to add a transitional content between the different learning atoms as well as context-specific atoms such as an overview, a motivation, and a summary for the whole course. In many cases it is also necessary to adapt the layout of some of the learning atoms to meet a selected standard layout for the whole course.

We are now in a position to complete Figure 2 (Figure 13).

Proof of concept

Project background

The work underlying the paper evolved as part of the five-year program ViKar (Virtual University Karlsruhe) that, among others, undertook to explore how to develop shared course material that could subsequently be adapted to the needs of various schools of higher learning with different educational goals. Our focus was less on content and more on methodical issues of how to prepare the individual courses. Hence, we simply used an already existing suite of one-semester database courses that were of interest to all participating schools. The student sample was relatively broad: Third-year computer science students of a university, second-year business school students of a university, third-year students of a polytechnical school, and third-year students who alternated between the school and work at industry. Consequently, there were considerable differences with regard to both the selection and depth of the teaching material, and the didactic strategies. Our approach evolved slowly as we tried to deal with the – sometimes sobering – experiences, often as mundane as differing notations and terminology, or incompatible examples for illustration and exercises.

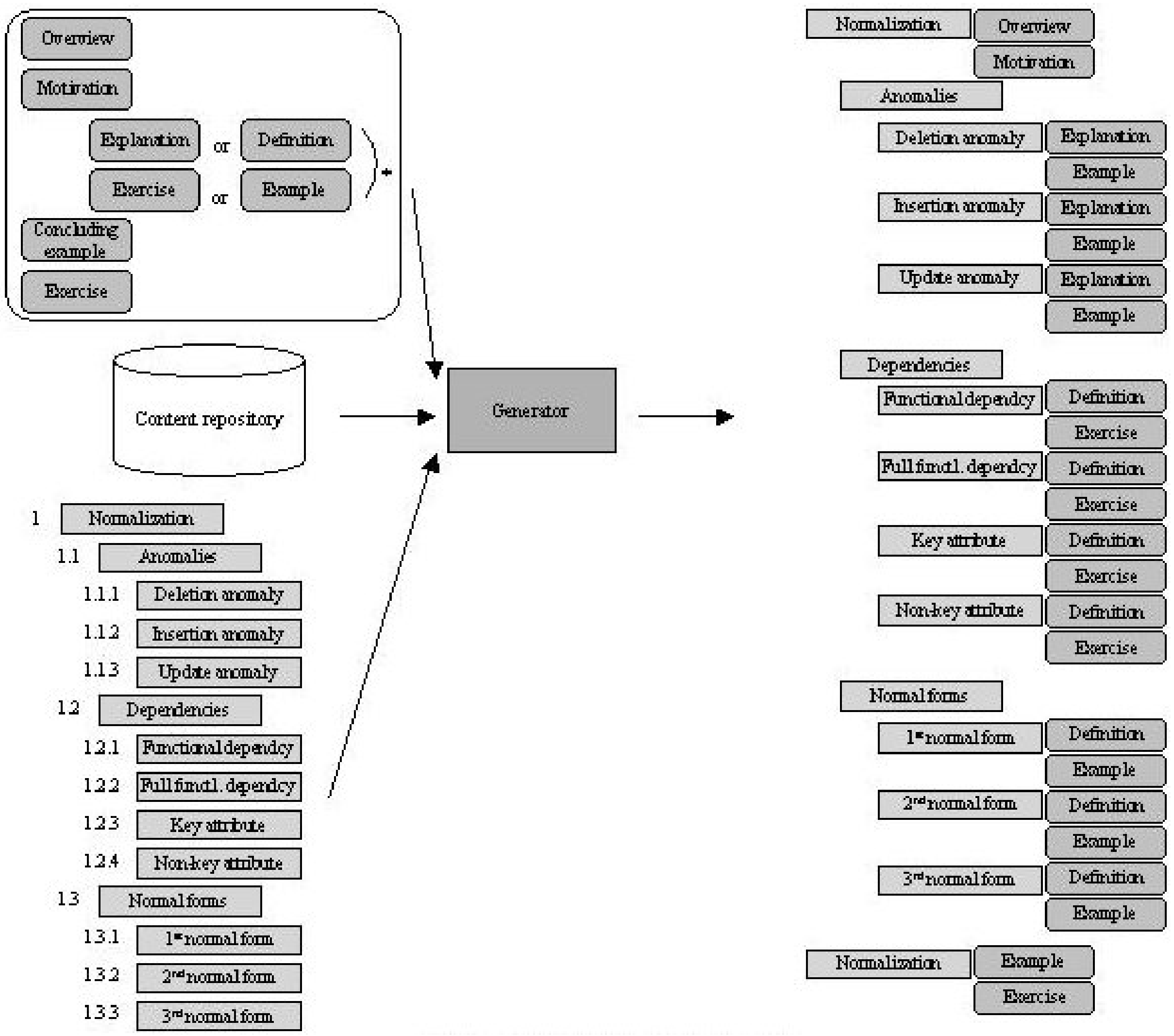


Figure 12. Weaving content and didactics: Example

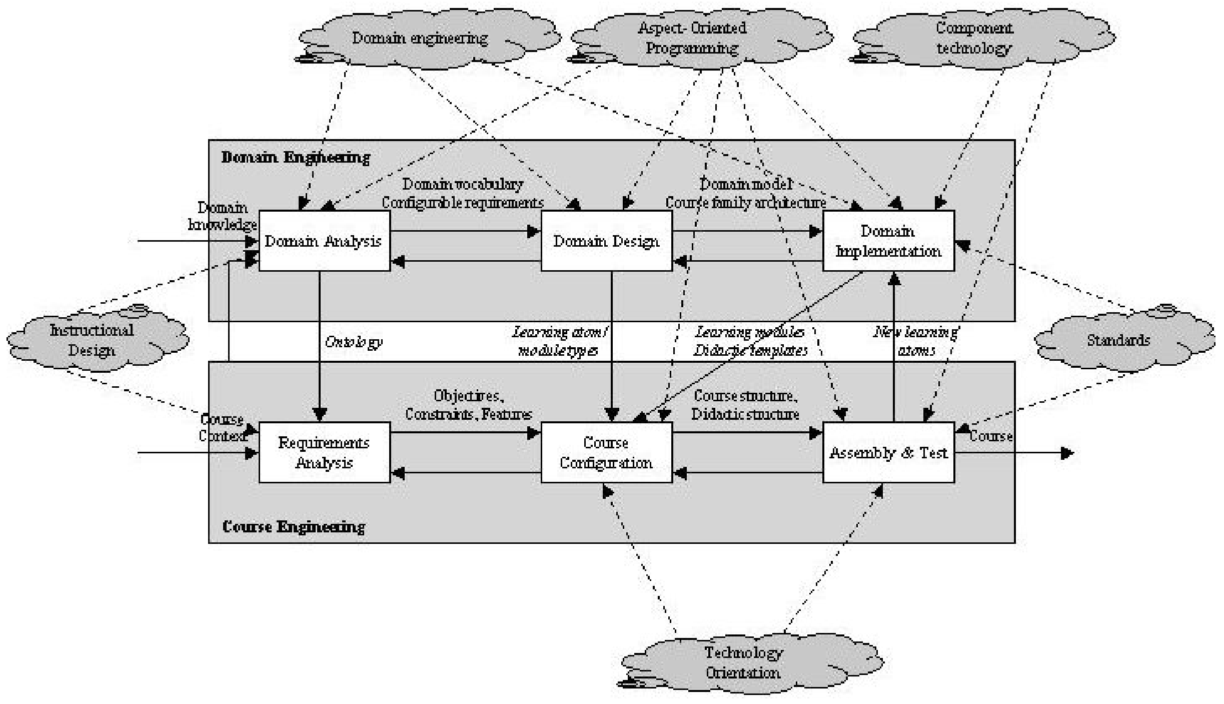


Figure 13. Figure 2 refined

Implementation

Much of the effort went into domain engineering. Over the years a complete ontology for database technology was developed, driven both by standard textbooks and the powerpoint presentation of a complete 16-week university course at 3 hours per week, augmented by course material from the other three schools. Once a significant portion of the ontology was available we were able to systematically determine a large set of learning objects (atoms and modules) for the university courses and selective sets for the other schools, and complete them using much of the existing material.

We could thus demonstrate that working from an ontology is indeed a viable approach to domain engineering. In particular, the approach is very well-structured and gives conciseness and unambiguity to the notion of learning object. We are certain that the same approach can be applied to all disciplines with a well-structured domain, such as the natural and engineering sciences.

Technical platform: SCORE

To support the experiments we developed a prototypical learning system, SCORE (System for COurseware REuse) (SCORE, 2004). SCORE distinguished between different participants of the learning/teaching process such as author, instructor, and student, and provides a suitable environment for each of these groups.

SCORE is based on standard technologies that support openness, portability, and reusability, particularly relational/XML databases (Oracle9.1), Java, J2EE and XML, learning technology standards such as LOM and IMS content packaging, and design patterns such as Three-Tier architecture and Model-View-Controller. Figure 14 shows the system architecture of SCORE with three layers:

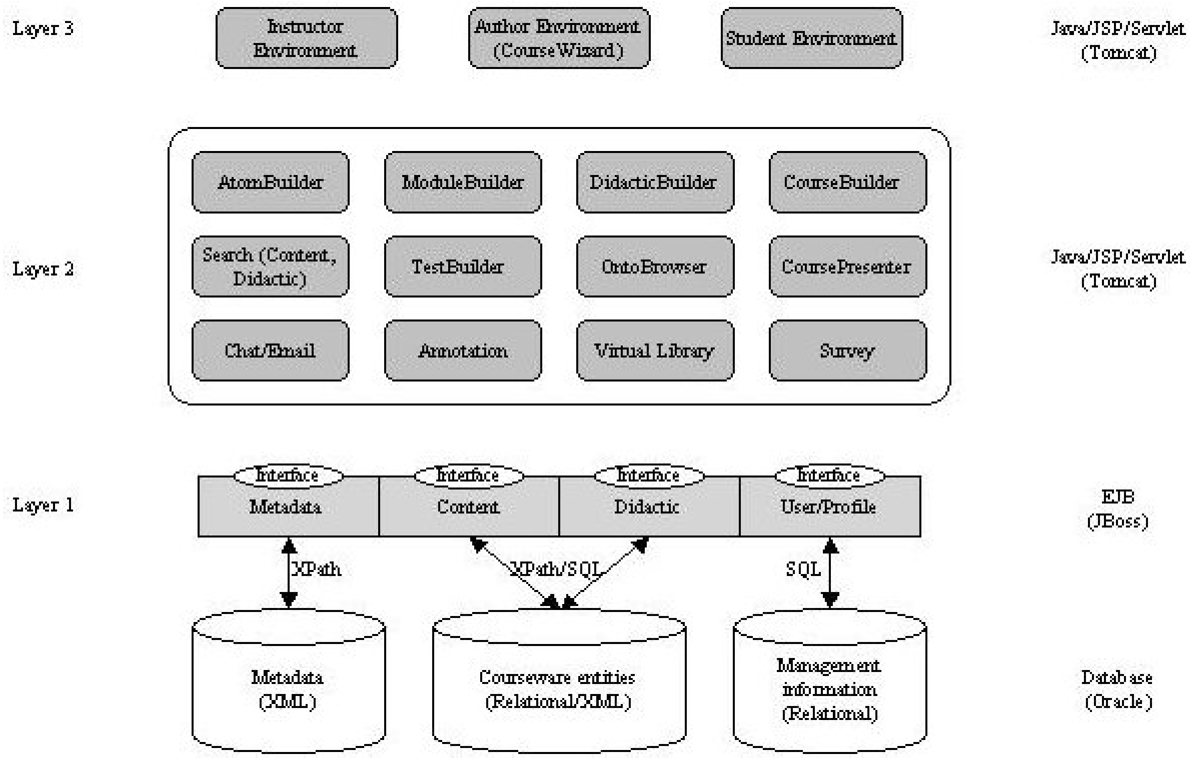


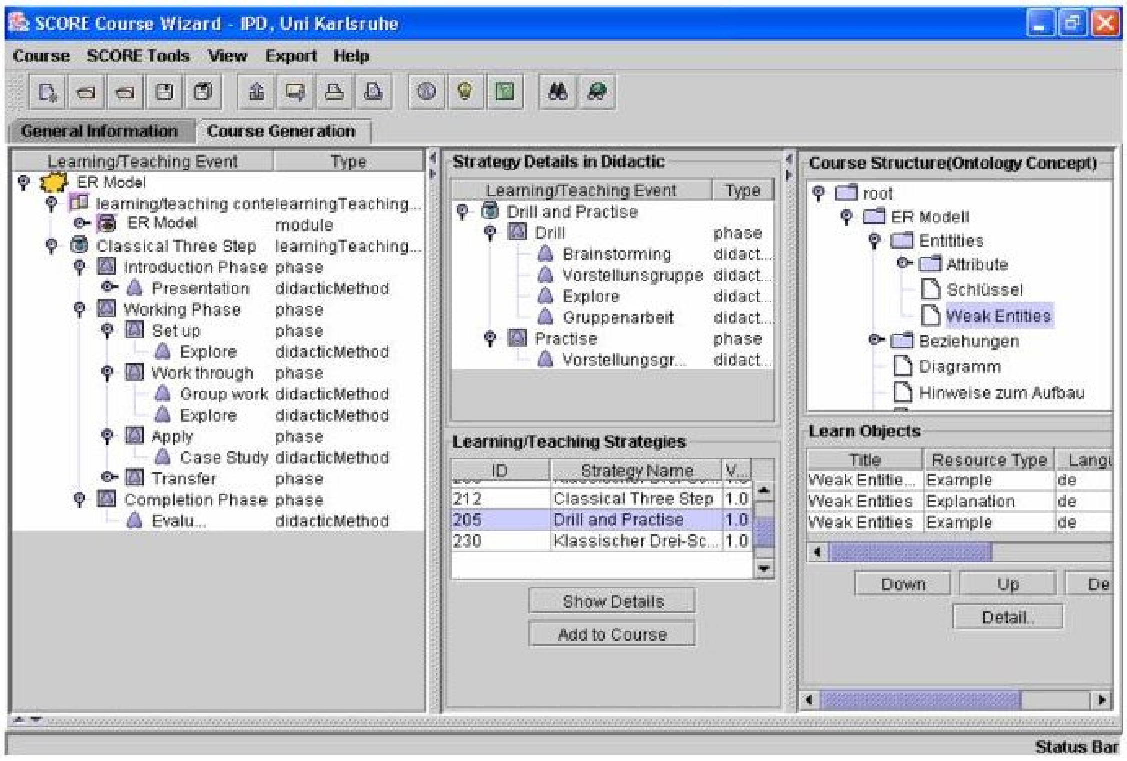
Figure 14. SCORE architecture

1. The data base and data access layer (layer 1) manages all types of data needed or produced during the processes of courseware development and reuse, learning, and teaching. In particular, this layer includes a courseware repository that manages the different kinds of courseware entities such as learning objects (learning atoms, modules, course module) and didactic entities (didactic methods, learning/teaching strategies, didactic templates).
2. The basic services layer (Layer 2): As the core of the SCORE system the layer provides a modular framework of common tools and services that are necessary for conducting different activities or tasks by the different user types. These tools or services are reusable building blocks used to assemble or build

complete user-specific (author, instructor, and student) environments at layer 3 (see below). The services or tools found in this layer can be classified into three categories:

- Courseware development tools. These tools are used in the course engineering process and help the developer to conduct the different activities of the development process in an efficient way. Examples for tools in this category are AtomBuilder, ModuleBuilder, DidacticBuilder, and CourseBuilder.
- Learning/teaching tools. These tools are used in the learning/teaching process. From the course engineering point of view, these tools are considered reusable technology components that can be combined to provide a suitable learning/teaching environment for a specific learning/teaching method and content. Examples for tools in this category are CoursePresenter, Annotation, Chat/Mail.
- General tools. These are tools that are used by the different user types. Examples for such tools are the OntoBrowser and ScoreSearch.

3. The user environments layer (layer 3): It includes specialized environments for each user type (author, instructor, and student). We restrict our discussion to the author environment as the one most relevant to this paper. The environment provides tools that allow one to effectively conduct the courseware development process. The components are integrated within a tool we call "Course Wizard (CW)".
   - The AtomBuilder is used to build a new learning atom. It provides a comfortable user interface for describing a learning atom by SCORE metatadata and assigning it to a learning topic (module) in the domain-specific ontology. For the production of the atoms themselves, the author is able to use any available content production tools.
   - The ModuleBuilder supports the course (content) engineering process. It is mainly used to deal with learning modules and provides all functionalities necessary for transforming a (general) learning module into a (learning context specific) course module. Therefore, the ModuleBuilder makes use of other tools such as the OntoBrowser and ScoreSearch.
   - The DidacticBuilder deals with the learning process didactic discussed. It is used to find, select, create, and reuse learning/teaching strategies and methods.
   - The CourseBuilder allows to build complete learning/teaching events (course, seminar, …). It is used to merge and integrate the different components of the different concerns to a complete learning/teaching event.
   - The TestBuilder supports the creation of tests for the students.



Learning/Teaching Event	Type
ER Model	
learning/teaching conte	learningTeaching...
ER Model	module
Classical Three Step	learningTeaching...
Introduction Phase	phase
Presentation	didacticMethod
Working Phase	phase
Set up	phase
Explore	didacticMethod
Work through	phase
Group work	didacticMethod
Explore	didacticMethod
Apply	phase
Case Study	didacticMethod
Transfer	phase
Completion Phase	phase
Evalu...	didacticMethod

Learning/Teaching Event	Type
Drill and Practise	
Drill	phase
Brainstorming	didact...
Vorstellunsgruppe	didact...
Explore	didact...
Gruppenarbeit	didact...
Practise	phase
Vorstellungsgr...	didact...

ID	Strategy Name	V...
212	Classical Three Step	1.0
205	Drill and Practise	1.0
230	Klassischer Drei-Sc...	1.0

Title	Resource Type	Langu
Weak Entitie...	Example	de
Weak Entities	Explanation	de
Weak Entities	Example	de

Figure 15 Course Wizard: Screenshot

Figure 15 shows a screen shot of the CW with, from right to left, the ModuleBuilder, the DidacticBuilder and the CourseBuilder. The screen shot nicely shows how the user interface reflects the aspect separation of content and didactic. A complete example of how to use the CW in practice can be found in (Ateyeh, 2004).

Evaluation and conclusions

After reviewing the state of the art, we set ourselves several goals. It should be possible to design courseware in small units, with a clear separation of the various aspects, in our case predominantly contents and didactics. Their interdependence should only be taken into account when courseware is produced for a specific learning/teaching context. There should be a clear separation of the modeling phase – corresponding to engineering for reuse – and the production phase – corresponding to engineering with reuse. The production phase should largely be automated, somewhat in the sense of model-driven programming. Our approach was modeled after the software engineering techniques of software reuse, component technology, and aspect-oriented programming.

The combination of these techniques poses challenges even under ordinary circumstances and has never before been applied to courseware engineering. As a first attempt in this direction we feel we have been quite successful. Content is the driving force. Through a domain engineering approach that is based on a commonly agreed ontology a first component structure is defined. Component adaptability is achieved by introducing the concept of learning resource type. It allows refining an ontology-based component in a large variety of ways that each suit a specific instructional intent. The resulting so-called learning atoms can then be implemented and maintained in a repository.

We also managed to separate the aspects of contents and didactics. We could show that one can derive a host of didactical methods for a given didactical strategy, and then detail the method in the form of a template that prescribes how atoms from the repository should be composed and ordered to meet the instructor's and/or student's objectives. A specific course can be generated almost automatically by a weaving process that takes a template as a prescription for which atoms to draw from the repository and how to arrange them in order.

Nonetheless, the results are only a first and still modest step towards the flexible and economical production of courseware for a large variety of settings. One weakness of our work is the dearth of empirical evidence on the suitability of our approach. The approach evolved slowly in the course of a five-year project on a virtual university. Originally the emphasis was on the content level by ontology engineering. During the course of the work it became apparent that the desired flexibility could only be achieved if content and didactic were dealt with separately. This resulted in the concepts of didactic template and aspect weaving by generation. Unfortunately, the short remaining time of the program did not allow us to test the strategy on a larger scale.

As a consequence, more empirical evidence must be gathered to convince oneself that the separation of content and didactic works under many conditions, and what these conditions are. Likewise, before undergoing the vast effort of building an ontology criteria would have to be developed whether a subject area is well-structured enough to justify the effort. As a first step, one should apply the approach to additional domains beyond the subject area of database courses. Also, for reasons of economy, it seems incumbent to integrate the SCORE courseware development and reuse environment into existing commercial learning platforms that support the learning technology standards and provide (to some extent) a modular learning/teaching environment.

We were not entirely successful to separate the aspects of contents and didactics. Although we did better than other current work, we still had to merge content and didactic principles – as judged from our goals – prematurely during the domain design phase. Future work should concentrate on how to shift more of the merging into course engineering.

Another characteristic of the approach is its heavy reliance on the existence of a suitable ontology. At least initially ontologies will have to be developed quite often, and this requires a fairly large investment in time and competent manpower and, therefore, there are considerable delays before the first course becomes available for a particular subject area. Consequently, one should steadily observe the market for ontology development tools.

Acknowledgment

The authors gratefully acknowledge the collaborative and stimulating environments of ViKar with Wolffried Stucky, Klaus Gremminger, Rudolf Krieger, Daniel Sommer, Jutta Mülle, and of SCORE with Birgitta König-Ries and Michael Klein.

References

ADL (Advanced Distributed Learning Initiative) (2004). Sharable Content Object Reference Model (SCORM): SCORM 2004 3rd Edition, retrieved May, 30, 2006 from http://www.adlnet.gov/scorm/downloads/index.cfm.

AOSD (2003). Aspect-Oriented Software development, retrieved May, 30, 2006 from http://aosd.net.

Aksit, M. (1989). *On the Design of the Object-Oriented Programming Language Sina*. Ph.D. thesis, University of Twente.

Ateyeh, K. & Klein, M. & König-Ries, B. & Mülle, J. (2003). A Practical Strategy the Modularization of Courseware Design. In: *Professionelles Wissensmanagement – Erfahrungen und Visionen: Adaptive E-Learning and Metadata*, Luzern.

Ateyeh, K. (2004). *Reuse-Driven Courseware Engineering*. Ph.D. thesis, Universität Karlsruhe. Shaker Verlag.

Bell, J., Bellegarde, F., Hook, J., Kieburtz, R. B., Kotov, A., Lewis, J., McKinney, L., Oliva, D. P., Sheard, T., Tong, L., Walton, L.& Zhou, T. (1994). Software Design for Reliability and Reuse: A Proof-of-Concept Demonstration. In *Proceedings of the Conference TRI-Ada*.

Biggerstaff, T. J. & Perlis, A. J. (1989). *Software Reusability, Concepts, and Models*. Vol. 1. ACM Press. Addison-Wesley.

Blumstengel, A. (1998). *Entwicklung hypermedialer Lernsysteme*. (Development of Hypermedia Learning Systems, in German) Ph.D. thesis, Universität Paderborn.

Boles, D. & Schlattmann, M. (1998). Multimedia-Autorensysteme: Grafisch-interaktive Werkzeuge zur Erstellung multimedialer Anwendungen. (Multimedia Author Systems: Graphical-interactive Tools for the Development of Multimedia Applications, in German) LOG IN 18 (1), 10-18.

Czarnecki, K. & Eisenecker, U. W. (2000). *Generative Programming*. Addison-Wesley.

Garzotto, P., Paolini, P. & Schwabe, D. (1991). HDM – a Model for the Design of Hypertext Applications. In *Proceedings of the 3rd Annual ACM Conference on Hypertext Systems*, 313-328.

Grützner, I., Pfahl, D. & Ruhe, G. (2002). Systematic Courseware Development Using an Integrated Engineering Style Method. In: *Networked Learning NL2002*, Berlin.

Henninger, S. (1997). An Evolutionary Approach to Constructing Effective Software Reuse Repositories. *ACM Trans. Software Engineering and Methodology (TOSEM*) 6 (2), 111-140.

Holsapple, C. W. & Joshi, K. D. (2002). A Collaborative Approach Ontology Design. Comm. ACM 45 (2).

IBM (2004). Subject-Oriented Programming, retrieved May, 11, 2006 from http://www.research.ibm.com/sop/.

IEEE Learning Technology Standards Committee (LTSC) P1484.1/D9 (2001). Draft Standard for Learning Technology Systems Architecture (LTSA), retrieved May, 30, 2006 from http://ltsc.ieee.org/wg1/files/IEEE_1484_01_D09_LTSA.pdf.

IEEE Learning Technology Standards Committee (LTSC) WG12 (2003). LOM Meta Data Standard. URL: http://ltsc.ieee.org/wg12/.

IMS (2004). IMS Content Packaging Information Model Version 1.1.3 Final Specification, retrieved May, 30, 2006 from http://www.imsglobal.org/content/packaging/index.cfm.

IMSLD (2004), retrieved May, 11, 2006 from http://www.imsglobal.org/learningdesign/index.html.

Isakowitz, T., Stohr, E. A. & Balasubramanian, P. (1995). RMM: A Methodology for Structured Hypermedia Design. Comm. ACM 38 (8).

Klein, M. (2002). *Courseware Engineering: Ein Vorgehensmodell zur Erstellung von wiederverwendbaren hypermedialen Kursen*. (Courseware Engineering: A Procedural Model for the Development of Reusable Hypermedia Courses, in German) Ph.D. thesis, Universität Karlsruhe.

Lieberherr, K. J. (1996). *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*. PWS Publishing Company.

Macromedia (2004a). Authorware 7, retrieved May, 11, 2006 from http://www.macromedia.com/software/authorware/.

Macromedia (2004b). Director MX 2004, retrieved May, 11, 2006 from http://www.macromedia.com/software/director/.

Merrill, M. D., Li, Z. & Jones, M. K. (1990). Second Generation Instructional Design. *Educational Technology* 30 (2), 7-14.

Merrill, M. D. (2000). First Principles of Instruction, retrieved May, 30, 2006 from http://projects.ict.usc.edu/itw/gel/MerrillFirstPrinciples02.pdf.

Pawlowski, J. M. (2001). *Das Essener-Lern-Modell (ELM): Ein Vorgehensmodell zur Entwicklung computerunterstützter Lernumgebungen*. (The Essener-Learning-Model (ELM): A Procedural Model for the Development of Computer-Supported Learning Environments, in German) Ph.D. thesis, Universität Essen.

SCORE (2004). SCORE Metadata, retrieved May, 11, 2006 from http://www.ipd.uka.de/SCORE/xsd/score_v1.xsd.

SEI (Software Engineering Institute) (2004): Component-Based Software Development. retrieved May, 30, 2006 from http://www.sei.cmu.edu/str/descriptions/cbsd.html.

Szyperski, C. (2003a). Component Technology: What, where, and how? In *Proceedings of the 25th International Conference for Software Engineering*, 684-693.

Szyperski, C. (2003b). Component Software: *Beyond Object-Oriented Programming*. 2nd ed. ACM Press.

Tennyson, R. D. & Rasch, M. (1995). Instructional System Development: The Fourth Generation. In: *Automating Instructional Design: Computer-Based Development and Delivery Tools*. Springer, 33-78.

ToolBook (2004). ToolBook, retrieved May, 11, 2006 from http://www.toolbook.com/

Uschold, M. & Gruninger, M. (1996). Ontologies, Principles, Methods and Applications. *Knowledge Engineering Review, 11* (2), 93-155.