SKIT, An Open Architecture for Courseware Authoring

Atze Dijkstra, Martijn Schrage and Doaitse Swierstra
{atze,martijn,doaitse}@cs.uu.nl
Department of Computer Science
Utrecht University
P.O. Box 80.089, 3508 TB
Utrecht, The Netherlands

Abstract - *Transfer of knowledge is an essential ingredient of the human way of living, necessary for survival in general but even more so in our information oriented society. Educational institutions make their best effort in offering knowledge, hoping that students will acquire this knowledge as efficiently as possible. The ways and means by which knowledge is transferred do not restrict themselves to the traditional ones anymore, but are becoming more varied with the growth of the variation of media capable of storing and transferring knowledge.*

Within this context, research is often focussed on the receiving side: how does a student (a receiver) receive knowledge, what is a good way of presenting it, given a certain model of learning? Other issues concern the different technologies available to implement an educational presentation for a receiver: text, images, sound, movie clips, and interactions together designated by the term multimedia. However, one issue is addressed less frequently, i.e. how do we as authors (the creators) of such educational presentations organize our view on the material. How is material organized, grouped; how is this material annotated with meta information normally only stored inside the mind of authors?

This paper argues that especially these last issues concerning the creating side are generally not well supported in existing courseware authoring systems. Writing courseware often involves writing a presentation to be used by students, but in the process of creating such a presentation the original insights, concerning design decisions is lost. We argue that in order to deal with the variations in the giving and receiving ends of knowledge transfer, as well as the transfer itself, a full-fledged development environment is needed.

Especially in the field of computer science, where education faces the difficulty of a rapidly evolving knowledge domain, a system that provides a road map on the existing material will help to reduce time and effort needed to keep course material up to date. Other advantages include more uniformity in produced educational material, by having the system provide templates for common patterns in the material. The meta information present in the system could also be used to create tailor made, possibly electronic, presentations of the material, taking into account student-specific properties and circumstances.

Such a development environment should support specific development oriented requirements like versioning and storing of created artifacts, but it should also cater for the specific courseware authoring needs. We will compare existing environments with a first prototype courseware authoring and management system that we built. The prototype system attempts to form a connection between a traditional database layer and a powerful, intuitive user interface environment for creating and managing large bodies of objects and relations between these objects. Using the results of this comparison, we propose an architecture supporting the separately usable abstractions of data (i.e. content) and relationships between data (i.e. structure).

Introduction

At the Department of Computer Science at Utrecht University we experienced that our functional programming courses did not achieve the effects on the students that we hoped for. In an effort to improve the learnability of these programming courses, we started a project named SKIT (short for Structuring Knowledge transfer using Information Technology) to reorganize the available course material. As part of this effort, we also started to make a tool - a courseware authoring system - which would support this reorganization.

In this paper, we first describe the requirements of such a courseware authoring system, followed by an examination of what is already available. Then we will have a look at our prototype system and the way educational material is modeled and manipulated. We will conclude with a short description of what - from our point of view - the architecture of a courseware authoring system should be.

Requirements of a courseware authoring system

Creation of material with the purpose of knowledge transfer involves the author and student as well as the material itself.

We will have a closer look at the requirements for these three aspects from the viewpoint of the author, that is, its relevance to the writer of material.

Author's aspects

The main task of an author is to organize material in such a way that consumption by a student is optimal. An author thus has to be able to organize material, for example in the form of a hierarchical overview of the material. At the same time, the author should not be restricted to a particular way of organizing. An author may have different views on material, for example in the form of a "required knowledge" relation imposed on the material. Material must be reusable, for example when constructing courses for a different audiences using the same material.

Besides the aspects of material directly visible in a publication for students, an author also uses other knowledge. An author may wish to administrate in what order material is offered to students. This order itself however might also be derived from other meta-aspects, for example the already mentioned "required knowledge" relation between material induces a (partial) order for presenting material to students. This meta-knowledge of material is often overlooked when material is created and re-used. However, if administrated, this meta-knowledge provides valuable information for an author as well as for co-authors.

More practical requirements concern the fact that a courseware authoring system does not function on its own. Existing material lives on normal files, often with the naming within a filesystem as the only explicitly visible structure. An author must be allowed to gradually move from such an existing environment to a system providing more features for structuring.

Authors often cooperate with others. This necessitates facilities for working together, in the form of import/export mechanisms or material sharing (with all the resulting simultaneous access related problems). The already mentioned administration of meta-knowledge also eases cooperation.

Student's aspects

A courseware authoring system primarily focusses on support for the author of material. However, this material is meant for students. An author may need to take student related aspects into account, for example, the amount of material a student can absorb over a fixed period of time. In general, an author may wish to present the material adapted to the needs and restrictions of students. This may involve theory about the way people learn as well as actual measured behavior (profiling) of students and the way material is used [22].

Aspects of material

Irrespective of the specific way data is used and interpreted, it has to be treated as something which evolves over time. Material is created, changed, (re)used and finally not used anymore. In short, it is treated as data in any development environment. As such, material exists in variants over time, necessitating version management. Material also is (re)used in different configurations; this necessitates configuration management.

Another aspect concerns the fact that material may often be available on different kinds of media as well as be encoded in different ways. An example may be available on file but also on the WWW and this example may take the form of a picture as well as plain text. With respect to this aspect, the system functions as a kind of gatherer of material, reorganizing and republishing it.

Consequences for "the system"

As a consequence of the aspects mentioned of the requirements we need a system supporting the construction of material using any source, where structure may be independently constructed. We will restrict ourselves to these aspects in this article. Other consequences of the preceding requirements are also important but not relevant for this article.

Existing courseware authoring systems

We have examined a number of other courseware authoring and management systems to see if they would fit our needs. Although we did keep our search limited to the World Wide Web, we have not found any systems that sufficiently met all criteria. Example systems we investigated are HyperCourseware [9] and Caleidoscope [7], and to a lesser extent the ACME [12] and Microcosm environments [14]. The EPOC initiative [4, 5, 21] tried to establish a standard for open courseware, but at the time of this writing has been discontinued.

Most of the systems are platform dependent and rely on the local file system for data storage. This puts a restriction on the kind of relations that can be defined on the data, and the granularity of these relations.

Another shortcoming is that existing systems tend to focus mainly on selecting and combining appropriate blocks of courseware and presenting these to students. However, we need a system, which helps us constructing these pieces of courseware. Ideally, this is done by instantiating standard educational patterns.

Finally, we would also like to be able to use the system for design and maintenance of conventional educational material, like readers, handouts and slide presentations. However, none of the examined systems offered any support for generating these textual presentations of material.

Existing general purpose development environments

Development environment features

Development environments allow the user to create artifacts. These artifacts are meant to be used in another environment where changing these artifacts is no longer possible. Many programs fall in the category "development environment", not only programs generating executable code but also editors (developing text).

Irrespective of the wide range of possible uses, these programs have several aspects in common: they administrate data for a user. Data evolves over time, is derived from other data, and depends for its correctness on other data. Support for these aspects, that is, version management, derivation management and configuration management, ideally is available for any piece of data produced. Support for these aspects should be factored out of any specific development environment and offered in a more general, systemwide fashion, preferably in an operating system.

Several attempts have been made to offer this common functionality in a general way [8, 10, 23]. We will consider some aspects of the Portable Common Tool Environment (PCTE) and Camera, especially those that are relevant for a courseware authoring system.

PCTE

PCTE, Portable Common Tool Environment [23] offers an elaborate infrastructure upon which development tools can be built. This infrastructure offers a platform independent interface for such tools as well as a framework for data, presentation and control integration. Though integration of all these aspects is equally important, we focus on data integration only.

PCTE offers its user a model of data that consists of objects and links between objects. An object is a set of named attributes. A link between two objects explicitly relates these two objects. Object attribute values have a type, and some attributes - like content - are obligatory. Links have implementation information associated with them, for example what to do with a linked target object if a source object is deleted. Links also have a name associated with them, allowing links to be traversed by name.

Different tools can arbitrarily create links for already existing objects, thereby creating a potential link spaghetti hazard. PCTE therefore offers mechanisms to create new views on objects and their links. These mechanisms allow views to be combined and filter out irrelevant links and attributes.

The typing system of PCTE is described using PCTE objects and links. Creating new types works object-oriented, i.e. using inheritance of attributes.

Camera

Basically, Camera [6, 10] offers the same model of data for a user as PCTE does. Objects with attributes can be used and defined in an object-oriented way. Objects can be related using relationships. These relationships are grouped in relations. Objects are not aware of being related. A relation does not incorporate implementation information like in PCTE.

On top of this basic data model other services are built into Camera. Version management is integrated into the system by means of snapshots of images of objects and relations, thereby offering consistent, versioned configurations of data.

Further services include derivation management in which dependencies between calculated values and their required objects are maintained and used as an equivalent for the UNIX *make* program.

Other systems

The basic idea of both PCTE and Camera is to offer an abstraction of the concept of data, thus allowing tools to share data available via the system. The resulting model with objects and a way to relate objects to each other is an idea that exists in many other tools, for example CASE tools supporting some object oriented analysis paradigm. UML [1] tools from Rational [16] explicitly model by way of classes and their associations. Relational databases (as described by e.g. [2, 3]) are another example in which attributes of objects are stored in tables. However, tuples in a relational database have no identity, thus necessitating explicit identifier management in order to link objects (mapped on tuples). Corba [15] also defines a relation service, where relationships between objects are administrated outside of the objects themselves.

File systems of current operating systems mostly offer objects with a fixed amount of attributes, for example content and owner. Structuring is also limited, only the hierarchy imposed by the naming service of a file system can be used to organize files.

Data model discussion

Why is this model of objects as identifiable aggregates of attributes of data the right one for a courseware authoring system? In general, the idea of data boils down to basic values (like strings), pointer values (to other values) and structures of values (aggregate values). These different categories of values are generally mixed; most object-oriented languages allow attributes of an object to be a combination of basic values and references to other objects. The resulting structure is formed according to invariants and patterns, (hopefully) only known by the associated code in the data model.

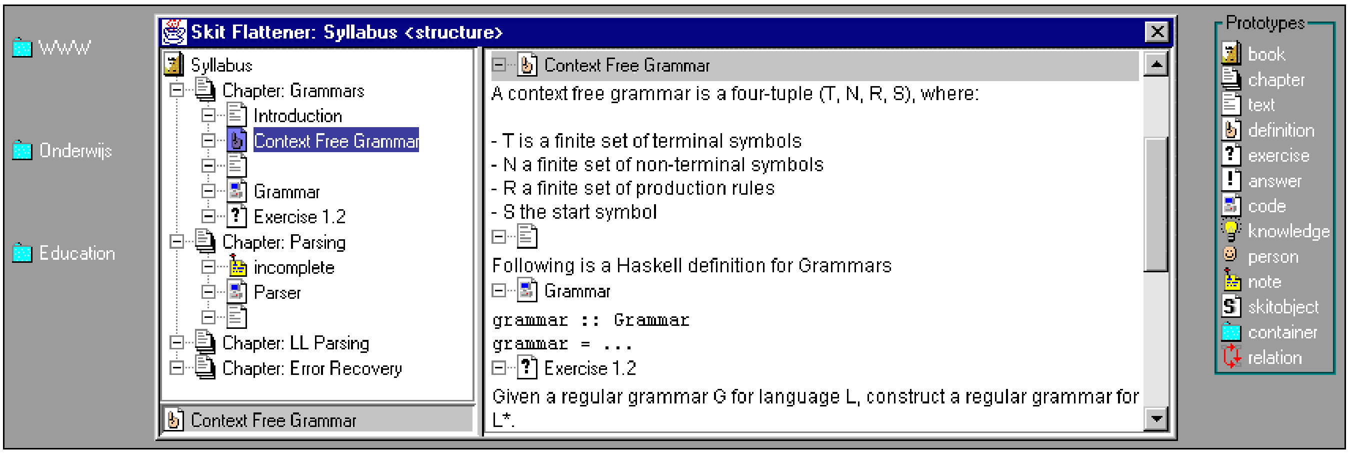

Figure 1: The Skit Flat Representation Editor or flattener.

A problem with this "combine basic value + pointers" approach is that new patterns of already existing objects needed for new tools cannot always be incorporated into existing class definitions. Code may simply not be available, or it would conflict with design heuristics [17] like "do not place unrelated behavior into one class". In a courseware authoring system this is undesired behavior, because material will be re-used by different tools.

Another way of looking at data might be the relational database view [3]. Data is put into attributes forming tuples, that is, aggregates of basic values. Tuples are grouped into tables. However, tuples are anonymous, that is, data has no identity as in an object-oriented framework. Identity and selection mechanisms have to be explicitly constructed since the purely relational view on data lacks an appropriate mechanism. However, creating specific (thus identifiable) structures of objects (thus needing pointing) is what will happen in any development system.

These observations, together with the requirements, lead to an architecture in which the data model as discussed above takes a central place. Before we further discuss this architecture, we will first show how material fitting this model can be used in a concrete prototype courseware system.

SKIT Prototype

Brief Overview

We have built a prototype system to test the completeness of our requirements, and determine a good architecture for the future system. For portability reasons the prototype has been implemented using Java. Seen from the user interface, the prototype resembles a file system, in which files can have arbitrary attributes. In contrast to a file system, however, the smallest identifiable objects in the system are not files containing documents, but typed units of educational material of these documents, like "example", "definition", and "exercise".

Another difference with a regular file system is that, instead of having only one hierarchical relation for organizing objects, an arbitrary number of relations can be used to organize the objects. In the system an object may participate in many different relations.

Relations are also used to impose structure on objects, for example to combine a collection of objects in a printable document. These structures can share objects between them, which facilitates writing documents that reason about program sources for example. The fragments of code that appear in the document can be shared with the actual source code, so no inconsistencies between the source and the document can occur.

Example

Figure 1 shows an actual screenshot of the system. It is a part of the desktop with the flat representation editor and a collection of object prototypes (which will be explained shortly). The objects, called SKIT objects or nodes, used in this example, are simplified pieces of educational material from the computer science course Grammars and Parsing. The structure they are part of determines how the syllabus of the course will be printed.

The left pane of the flattener contains a tree browser that can be used to view and edit relations over SKIT objects. The SKIT objects have a uniform appearance, and can always be used for drag operations, wherever they appear. They also support a context sensitive pop up menu that, apart from basic functions like move, rename, and edit operations, can contain additional menu items depending on the type of the object, or the browser it is being viewed in.

The icons in the box on the right side of the figure are object prototypes. They provide a mechanism to instantiate new SKIT objects by making use of the drag and drop mechanism. When they are dragged to a destination they are not actually moved, but a new object of that type is instantiated and dropped on the destination.

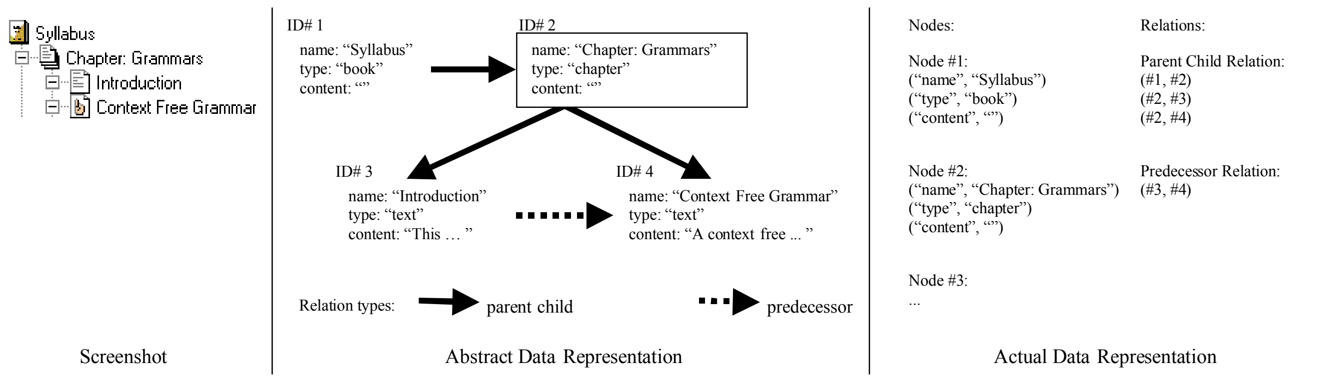

Figure 2: Data Representation.

The right pane of the flattener window contains a flat and editable representation of the structure in the left pane. Although it looks like a normal text document, it is actually a combination of the contents of the SKIT objects on the left side. All edit actions on the flattened version are propagated to the objects whose content is changed and, vice versa, when an object’s content changes, this is reflected in the flattener.

The flattener also serves as an outline editor; the structure of the document can be viewed and edited in the tree browser pane, and expanding and collapsing nodes in the tree causes the editor to show and hide, respectively, corresponding parts of the flat representation.

In addition to on-screen flattening, the system supports exporting data to files using a similar flattening mechanism. This so-called external file flattener also allows type specific transformations on SKIT objects. This way different presentations can be generated from the same SKIT objects. Currently, only LaTeX and Haskell code can be generated, but future versions will include html and support user defined formats.

Data Representation

Figure 2 contains a small part of the syllabus structure from Figure 1. The left part of the figure shows the structural relation as it is visualized by the system, the middle part shows the data structures underlying this visualization, and the right side shows how these data structures are implemented in the system.

Each SKIT object is implemented as a node (rectangles in Figure 2), which is an identifiable collection of name value pairs (attributes). The SKIT relation is implemented as a pair of relations: a parent child relation, denoted with the normal arrows, and a predecessor relation that determines the order of the children of each parent (denoted with dotted arrows). Relations are collections of relationships, which are the single arrows in the figure. Each relationship links two objects together, at least in a binary relation. The data layer also supports relations of other arities but they have not been used yet. There is actually a third relation involved, which links each parent to a corresponding predecessor relation, but it has been omitted from the picture for simplicity.

The actual data representation shows the clear separation between nodes and relations. Instead of using pointers that are part of the nodes, the relations are implemented using tables that contain references to the nodes. Each relation is a list of pairs (relationships) of SKIT object identifiers. The nodes are identified using a scheme of globally unique identifiers. Relations too are identifiable objects in the system that can be included in relations.

SKIT Architecture

Using the prototype and the preceding discussion about existing systems as a reference point we propose an architecture that will - in our eyes - fulfill the needs of a courseware authoring system.

On the most global level the SKIT system consists of four layers, as shown in Figure 3.

Applications

Tools & Facilities

Data

Storage

Figure 3: High level layering in the system.

The Data layer provides an abstraction of data in the form of objects and links between them. This layer provides the same functionality as PCTE and Camera concerning data. Data is put into uniquely identifiable aggregates of name + value pairs, called Nodes. These name + value pairs, or attributes, may not be other Nodes, that is, they may not point to other Nodes. Relations on the other hand consist of relationships where fields of a relationship may contain Nodes or other values. A relationship explicitly relates Nodes. An example of the way Nodes and Relations are used is shown in Figure 2. This part of the architecture has been realized in the prototype.

The Storage layer provides persistency of Nodes using different media. Persistence may be realized by a relational database but also via a set of HTML pages, a file system or a custom built database. Together the Data and Storage layers provide an abstraction of data and independence of the way it is stored. Currently a simple database is used.

The Tools & Facilities layer provides functionality for the integration of control and presentation, that is, building blocks to add new tools and their visualization with the facilities to have interaction between tools. The techniques to be used here are similar to those used in component technology like OpenDoc [11, 20], (D)COM [13, 20] and JavaBeans [19].

The Applications layer consists of the set of tools written using the Tools & Facilities layer. In the prototype the two upper layers have been combined and are simply part of the prototype itself. The application supporting the construction of the example syllabus and the required tools have not yet been separated into a framework and an application using this framework.

Future work

Currently the prototype as described above has been realized. The SKIT project is likely to be continued in a broader context where the system will be used in combination with tools to present material to students. For future research educational aspects as well as software architecture issues are relevant. The system will be used for constructing educational material as well as experiments with new tools for constructing this material. These tools will be built within a framework using component technology and domain specific languages.

In the near future, the prototype as well as architecture will be developed further in order to get a better idea about the long-term requirements. For more information contact the authors or have a look at the SKIT project WWW page [18].

We also expect that a tool developed along the proposed lines can be used for other problem domains, for example the writing of documentation or this paper itself.

References

1. Booch, Grady, Rumbaugh, James and Jacobson, Ivar, 'The Unified Modeling Language User Guide,' 1999.
2. Chen, Peter, 'Entity-Relationship Approach to Data Modelling,' Tutorial: System and Software Requirements Engineering, 1990, pp. 238-243.
3. Elmasri, Ramez and Navathe, Shamkant B., 'Fundamentals of Database Systems,' 1994.
4. 'EPOC Report 2, Pre-requisites for Open Courseware,' http://www.amtp.cam.ac.uk/icrd/EPOC/reports/epocrep2/epocrep2.htm.
5. 'Towards a Framework for Open Courseware. The third Report of the TLTP working group on open Courseware.,' http://www.amtp.cam.ac.uk/icrd/EPOC/reports/epocrep3/index.htm, 1996.
6. Florijn, Gert, Lippe, Ernst, Dijkstra, Atze, Oosterom, Norbert van, Swierstra, Doaitse, 'Camera: Cooperation in Open Distributed Environments,' Proceedings of the EurOpen and USENIX Spring 1992 Workshop/Conference, apr 1992, pp. 123-136.
7. 'Freebird Learning Systems - Caleidoscope,' http://www.coacs.com/Freebird/caleidos.htm.
8. Godard, C. and Charoy, F., 'Databases for Software Engineering,' 1994.
9. 'HyperCourseware, Solutions and Designs for the Electronic Educational Environment,' http://www.hypercourseware.com/.
10. Lippe, E., 'CAMERA, Support for Distributed Cooperative Work,' 1992.
11. MacBride, rew and Susser, Joshua, 'Byte Guide to OpenDoc,' 1996.
12. MacDougall, G. , Place, C. , Muldner, M., Currie, D., Khwaja, A., 'Automated Courseware Management Environment,' http://plato.acadiau.ca/sandbox/papers/calgary/acme1.html.
13. 'The Component Object Model Specification,' http://msdn.microsoft.com/developer/, 1995.
14. 'Microcosm,' http://www.mmrg.ecs.soton.ac.uk/projects/microcosm.html, 1997.
15. 'The Common Object Request Broker: Architecture and Specification,' http://www.omg.org, 1998.
16. 'Rational Software: Unified Development Solutions & Programming Tools,' http://www.rational.com/.
17. Riel, Arthur J., 'Object-Oriented Design Heuristics,' 1996.
18. Schrage, M. and Dijkstra, A., 'SKIT Project,' http://www.cs.uu.nl/docs/skit.
19. 'JavaBeans API Specification,' http://java.sun.com/beans/, 1997.
20. Szyperski, Clemens, 'Component Software,' 1997.
21. TLTP Working group on Open Courseware, 'EPOC Report 2, Pre-requisites for Open Courseware,' http://www.amtp.cam.ac.uk/icrd/EPOC/reports/epocrep2/epocrep2.htm.
22. Verpoorten, J.H., 'Modelgebaseerde Ontwikkeling van Computerondersteunend Onderwijs,' 1994.
23. Wakeman, Lois and Jowett, Jonathan, 'PCTE, The standard for open repositories,' 1993.