

# Highly Interactive Web-Based Courseware

**Dissertation**

der Fakultät für Informations- und Kognitionswissenschaften  
der Eberhard-Karls-Universität Tübingen  
zur Erlangung des Grades eines  
Doktors der Naturwissenschaften  
(Dr. rer. nat.)

vorgelegt von  
**Dipl.-Inform. Frank Hanisch**  
aus Reutlingen

**Tübingen**  
**2004**

Tag der mündlichen Qualifikation: 11.02.2004

Dekan: Prof. Dr. Martin Hautzinger

1. Berichterstatter: Prof. Dr.-Ing. Dr.-Ing. E.h. Wolfgang Straßer

2. Berichterstatter: O.Univ.-Prof. Dr.Dr.h.c.mult. Hermann Maurer  
(Technische Universität Graz)

3. Berichterstatter: Prof. Dr. Dr. Friedrich W. Hesse

## Zusammenfassung

*Zukünftige Lehr-/Lernprogramme sollen als vernetzte Systeme die Lernenden befähigen, Lerninhalte zu erforschen und zu konstruieren, sowie Verständnisschwierigkeiten und Gedanken in der Lehr-/Lerngemeinschaft zu kommunizieren. Lehrmaterial soll dabei in digitale Lernobjekte übergeführt, kollaborativ von Programmierern, Pädagogen und Designern entwickelt und in einer Datenbank archiviert werden, um von Lehrern und Lernenden eingesetzt, angepasst und weiterentwickelt zu werden. Den ersten Schritt in diese Richtung machte die Lerntechnologie, indem sie Wiederverwendbarkeit und Kompatibilität für hypermediale Kurse spezifizierte. Ein größeres Maß an Interaktivität wird bisher allerdings noch nicht in Betracht gezogen. Jedes interaktive Lernobjekt wird als autonome Hypermedia-Einheit angesehen, aufwändig in der Erstellung, und weder mehrstufig verschränk- noch anpassbar, oder gar adäquat spezifizierbar. Dynamische Eigenschaften, Aussehen und Verhalten sind fest vorgegeben.*

*Die vorgestellte Arbeit konzipiert und realisiert Lerntechnologie für hypermediale Kurse unter besonderer Berücksichtigung hochgradig interaktiver Lernobjekte. Innovativ ist dabei zunächst die mehrstufige, komponenten-basierte Technologie, die verschiedenste strukturelle Abstufungen von kompletten Lernobjekten und Werkzeugsätzen bis hin zu Basiskomponenten und Skripten, einzelnen Programmanweisungen, erlaubt. Zweitens erweitert die vorgeschlagene Methodik Kollaboration und individuelle Anpassung seitens der Teilnehmer eines hypermedialen Kurses auf die Software-Ebene. Komponenten werden zu verknüpfbaren Hypermedia-Objekten, die in der Kursdatenbank verwaltet und von allen Kursteilnehmern bewertet, mit Anmerkungen versehen und modifiziert werden.*

*Neben einer detaillierten Beschreibung der Lerntechnologie und Entwurfsmuster für interaktive Lernobjekte sowie verwandte hypermediale Kurse wird der Begriff der Interaktivität verdeutlicht, indem eine kombinierte technologische und symbolische Definition von Interaktionsgraden vorgestellt und daraus ein visuelles Skriptschema abgeleitet wird, welches Funktionalität übertragbar macht. Weiterhin wird die Evolution von Hypermedia und Lehr-/Lernprogrammen besprochen, um wesentliche Techniken für interaktive, hypermediale Kurse auszuwählen. Die vorgeschlagene Architektur unterstützt mehrsprachige, alternative Inhalte, bietet konsistente Referenzen und ist leicht zu pflegen, und besitzt selbst für interaktive Inhalte Online-Assistenten. Der Einsatz hochgradiger Interaktivität in Lehr-/Lernprogrammen wird mit hypermedialen Kursen im Bereich der Computergraphik illustriert.*



## Abstract

*The grand vision of educational software is that of a networked system enabling the learner to explore, discover, and construct subject matters and communicate problems and ideas with other community members. Educational material is transformed into reusable learning objects, created collaboratively by developers, educators, and designers, preserved in a digital library, and utilized, adapted, and evolved by educators and learners. Recent advances in learning technology specified reusability and interoperability in Web-based courseware. However, great interactivity is not yet considered. Each interactive learning object represents an autonomous hypermedia entity, laborious to create, impossible to interlink and to adapt in a graduated manner, and hard to specify. Dynamic attributes, the look and feel, and functionality are predefined.*

*This work designs and realizes learning technology for Web-based courseware with special regard to highly interactive learning objects. The innovative aspect initially lies in the multi-level, component-based technology providing a graduated structuring. Components range from complex learning objects to toolkits to primitive components and scripts. Secondly, the proposed methodologies extend community support in Web-based courseware – collaboration and personalization – to the software layer. Components become linkable hypermedia objects and part of the courseware repository, rated, annotated, and modified by all community members.*

*In addition to a detailed description of technology and design patterns for interactive learning objects and matching Web-based courseware, the thesis clarifies the denotation of interactivity in educational software formulating combined levels of technological and symbolical interactivity, and deduces a visual scripting metaphor for transporting functionality. Further, it reviews the evolution of hypermedia and educational software to extract substantial techniques for interactive Web-based courseware. The proposed framework supports multilingual, alternative content, provides link consistency and easy maintenance, and includes state-driven online wizards also for interactive content. The impact of great interactivity in educational software is illustrated with courseware in the Computer Graphics domain.*



## Acknowledgements

This thesis was conducted during my occupation as Research Assistant at the Graphical-Interactive Systems group of the Wilhelm Schickard Institute (WSI/GRIS), University of Tübingen.

First and foremost, I would like to thank my advisor Prof. Wolfgang Straßer for supporting me throughout my work, and giving me the freedom in research I needed. The referees Prof. Hermann Maurer and Prof. Friedrich W. Hesse I would like to thank for rendering their expert opinion with respect to my work.

Much gratitude is given to those who supported me and were of great help throughout the cooperations and meetings. Special thanks go to, in alphabetic order, Prof. Steve Cunningham (California State University Stanislaus), Dr. L. Miguel Encarnação (Fraunhofer CRCG, Providence), Prof. Dr. Thomas Ertl (University of Stuttgart), Prof. Dr. James D. Foley (Georgia Institute of Technology), Prof. Maike Franzen (University of Applied Sciences Solothurn), Prof. Mike McGrath (Colorado School of Mines), Prof. Dr. Michael H. W. Hoffmann (University of Ulm), Prof. Dr. Reinhard Klein (University of Bonn), and Prof. Dr. José Carlos Teixeira (University of Coimbra). I would also like to thank Dr. Beatriz Barquero (KMRC Tübingen), Tobias Bolch (University of Erlangen), Dr. Ulrike Creß (KMRC Tübingen), Prof. Craig Gotsman (Technion – Israel Institute of Technology), Dr. Kurt Kratschmann (IBM IT Education Services), Martin Rotard (University of Stuttgart), and Simon Wiest (University of Tübingen) for technical feedback and evaluation.

Further thank is due to the colleagues and students of the WSI/GRIS, especially to my students Sven Gottwald, Christian Holzer, Patrizia Nardin, and Benjamin Nill. The Runge Kutta integrator is courtesy of my colleague Michael Hauth, and the Shear Warp factorization of Alexander Ehlert.

A special thank belongs to my wife and my family for their appreciation.

# Contents

<b>1</b>	<b>Introduction .....</b>	<b>11</b>
1.1	Grand Challenges .....	11
1.2	The Computer Revolution .....	13
1.3	Structure and Scope .....	14
<b>2</b>	<b>Basics .....</b>	<b>17</b>
2.1	Interactivity .....	18
2.1.1	GUI Characteristics .....	18
2.1.2	Perception and Cognition.....	20
2.1.3	A Qualitative Framework .....	22
2.2	Hypermedia.....	24
2.2.1	Origins .....	24
2.2.2	Design Principles.....	25
2.2.3	The Web .....	29
2.3	Web-Based Courseware.....	31
2.3.1	Educational Software .....	32
2.3.2	Learning Management Systems.....	36
2.3.3	Content Management.....	38
2.3.4	Learning Technology Standards.....	42
2.4	Interactive Learning Objects .....	45
2.4.1	Repositories .....	45
2.4.2	Software Components .....	52
2.4.3	Scripting.....	54
2.5	Conclusion.....	57
<b>3</b>	<b>GRIS/ILO Interactive Learning Objects .....</b>	<b>59</b>
3.1	MVC Interactivity.....	60
3.2	Software Components.....	61
3.2.1	The ORC-SG Design Pattern .....	61
3.2.2	Object, Renderer, Constraint (ORC) .....	62
3.2.3	Scene Graph and GUI (SG).....	65
3.2.4	The Toolkit.....	70
3.3	Adaptability and Interoperability.....	71
3.3.1	Scripting.....	71
3.3.2	Networking .....	75
3.3.3	Scripting Database.....	77
3.3.4	Drag & Drop Scripting .....	79



<b>4 Web Framework.....</b>	<b>83</b>
4.1 Organization and Production.....	84
4.1.1 Layered Database Model.....	84
4.1.2 Template-Driven Generator.....	85
4.1.3 Offline Management.....	87
4.2 Web-based Authoring.....	89
4.2.1 Online Wizards.....	90
4.2.2 Learner Support.....	92
4.2.3 Author Support.....	95
<b>5 Case Studies.....</b>	<b>99</b>
5.1 Electronic Webmaster.....	100
5.1.1 Project.....	100
5.1.2 Institution.....	100
5.1.3 Bibliography.....	103
5.2 Image Processing and Video Communications.....	105
5.2.1 Project.....	105
5.2.2 Visual Programming.....	105
5.2.3 Component Programming.....	108
5.3 Scientific Visualization.....	110
5.3.1 Project.....	110
5.3.2 Scripting.....	110
5.3.3 Community.....	114
<b>6 Conclusion &amp; Directions for Future Work.....</b>	<b>117</b>
<b>Abbreviations.....</b>	<b>121</b>
<b>Name Index.....</b>	<b>122</b>
<b>Subject Index.....</b>	<b>124</b>
<b>Bibliography.....</b>	<b>127</b>



# 1 Introduction

## 1.1 Grand Challenges

In 2002, the *Computing Research Association* (CRA) asked what are the "grand research challenges" in computer science and engineering? Not surprisingly, education was selected as one of the big five [CRA02]. It is the 20-year old vision of information technology enabling learners to participate in community networks, where they engage with other learners, tutors, and teachers in self-expression, exploration, and learning by discovery and by doing. It is also the vision of a learning environment that adapts to the participant's needs in a transparent manner, and allows all participants to enhance this environment through the construction of both new learning objects and compositions of interoperable ones.

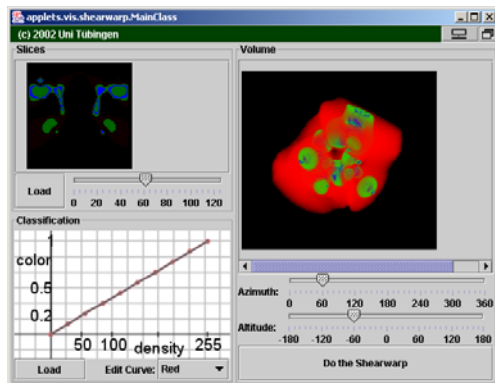
Keynote speeches at *ED-MEDIA 2002* [Barker02] echo the same longing for a system that lets children of all ages express, consider, and work with ideas of science (*Alan Kay*, "The Computer Revolution Hasn't Happened Yet"), a "Next-Generation Educational Software" (*Andries van Dam*) for creating dynamic science content based on both a pedagogical methodology – well-proven learner-driven and model-based techniques – and a multi-level, component-based technology. Educational material is meant to become reused in multiple contexts and for multiple purposes, and become organized and shared in digital libraries. A systematic reuse of such learning objects requires standardized metadata and presentation (*Hermann Maurer*, "What Have We Learned in 15 Years about Educational Multimedia?"); standards, in turn, could provide the basis for innovative collaborative learning techniques.

Research in learning technology currently specifies learning object metadata and learning management systems. Despite the fact that today's educational software is Web-based, and, from the very beginnings in the 1950s, has its strengths originating in its interactive, multimedia nature, research continues to focus on low-interactive, rather text-based material. While necessary, metadata does not meet our needs for interactive (dynamic) learning objects. How can we specify, link, adapt, exchange, and combine parts of interactive learning objects? Can we enable all community members performing these tasks? Instead, the "killer problem" [vanDam02] can be rather located in learning object design that aims towards gradation and interoperability. Research must develop adequate design principles allowing learners and educators to work with interactive material, adapting it to their needs, and integrating it in their environment. This thesis addresses these challenges and proposes adequate methodologies which are integrated into real educational software.

The thesis consists of two parts. At first, we review basic principles of interactivity, hypermedia, and educational software, and present our ideas of how Web-based teaching with interactive learning objects could (and should) be. Following an understanding of the concept of highly interactive Web-based courseware, we move on to the second part, where we propose and realize an adequate framework for it. We provide combined levels of

technological and symbolical interactivity with “Model View Controller (MVC) Interactivity”, and accompany visual programming with innovative visual scripting in the form of image-based “Drag & Drop Scripting”. Our “ORC-SG (Object, Renderer, Constraint, Scene Graph, and Graphical User Interface)” design pattern encapsulates matters of learning object state, appearance, functionality, and graphical visualization/interaction into reusable software components. Finally, the proposed Web framework introduces a “Layered Database Model” supporting multilingual, alternative content, a template-driven courseware generator assuring consistent linking and easy maintenance, and online wizards benefiting from an integrated state machine which offers authorization, session management, default values, undo, and preview.

We develop educational software in the field of Computer Graphics, which is archetypical in several aspects; we naturally face the needs for complex visualizations, we bring along profound knowledge in human-computer interaction, and we are familiar with component-based technology (see Figure 1). We started to employ interactive Web-based courseware at the department Graphical-Interactive Systems, Wilhelm Schickard Institute (WSI/GRIS), University of Tübingen, in 1995. Courseware typically accompanies lecture, and serves both as learning and programming platform for exercises and student projects. Apart from lectures on Computer Graphics (“Computergraphik spielend lernen”, awarded with a “Landeslehrpreis”), Computational Geometry, Geometric Modeling, Image Processing, Video Communications, and Scientific Visualization (“Spielend Visualisieren”), we performed student projects on Geometry, Cultural Heritage, and Dynamic Systems, and applied our courseware at Technion – Israel Institute of Technology, FernUniversität Hagen, University of



**Figure 1:** The shear warp factorization represents a volume rendering technique for visualizing 3D arrays of sampled data [Lacroute94]. This learning object reuses software components to visualize slices (top left) and offers interaction with the classification functions (bottom left) and the perspective parameters (right side).

Stuttgart, University of Erlangen, and IBM IT Education Services. Currently, we are participating in the foundation of the SIGGRAPH/Eurographics *CGEMS* (Computer Graphics Educational Materials) repository [Figueiredo03].

Throughout the thesis, we illustrate our ideas with extracts from our latest projects. While our department’s “Electronic Webmaster” demonstrates capabilities of the Web framework, two lectures on image processing and video processing illustrate how we apply model-based (top-down) visual programming and a low-level (bottom-up) component programming for education. Our courseware on scientific visualization further provides fine-grained hypermedia interlinking, visual scripting, and

community support. The “GRIS/ILO Interactive Learning Objects” repository features a representative cross section from more than 130 learning objects, available at <http://www.gris.uni-tuebingen.de/projects/ilo>.

## 1.2 The Computer Revolution

Web-based teaching is an interdisciplinary field. Terminology varies widely depending on the user’s background. Most evident is the denotation of interactivity in education, which has become an often-cited, long-winded term. We have combined it with two other magic bullets, hypermedia and courseware. What exactly is “interactive Web-based courseware”? Can we imagine education based on interactive hypermedia, that is, software enabling learners to create and evolve dynamic models to invent, express, and refine ideas? If so, what characteristics do we consider best? Let us review potential benefits of computer technology for facilitating learning as enumerated by *Alan Kay* [Kay91] in 1991.

*The first benefit is great interactivity. [...] A second value is the ability of the computers to become any and all existing media [...].*

*Third, and more important, information can be presented from many different perspectives. Marvin L. Minsky of MIT likes to say that you do not understand anything until you understand it in more than one way. Computers can be programmed so that “facts” retrieved in one window on a screen will automatically cause supporting and opposing arguments to be retrieved in a halo of surrounding windows. An idea can be shown in prose, as an image [...]*

*Fourth, the heart of computing is building a dynamic model of an idea through simulation. Computers can go beyond static representations that can at best argue; they can deliver sprightly simulations that portray and test conflicting theories [...]*

*A fifth benefit is that computers can be engineered to be reflective. [...] Finally, pervasively networked computers will soon become a universal library, the age-old dream of those who love knowledge. Resources now beyond individual means [...] will be potentially accessible to anyone.”*

Kay predicted the computer to become a direct manipulation learning tool, featuring great interactivity, proper use of hypermedia, and fruitful collaboration (see also his *Dynabook* vision [Kay77]). In the course of our argumentation, we will sharpen these concepts as follows:

- An interaction in educational software represents a learning process occurring while modifying objects (2.1.3). We consider an object as **highly interactive** learning object, if it provides means for manipulating its appearance, its dynamic state, and its functionality (3.1) directly by physical actions. Effects are immediately visible (2.4.2).
- Real **hypermedia** integrates interactive multimedia, means for collaboration, and means for personalization (2.2). Hypermedia linking must be reliable and fine-grained even for interactive objects (2.4.3). Today, educational software is Web-based (2.3.2).
- Sophisticated **courseware** facilitates administration, learning, and authoring (2.3.2). Content represents reusable, interoperable learning objects, archived in a repository, and shared, annotated, and expanded by the courseware community (2.3.3).

### 1.3 Structure and Scope

Let us briefly discuss the structure and scope of the thesis. Cross-references are enclosed in round brackets, indices for inline keywords (bold-faced), names (in italics), and references can be found at the end of the thesis.

The first part of the thesis reviews **basic principles** of interactivity (2.1), hypermedia (2.2), and Web-based courseware (2.3) with special respect to interactive learning objects (2.4).

We sharpen the term interactivity and emphasize its relevance for learning by identifying three approaches, the developer's view (2.1.1), the educator's view (2.1.2), and the communication theorist's view (2.1.3). Next, we reflect on characteristics and shortcomings of interactive hypermedia. We outline hypermedia's origins (2.2.1) and design principles (2.2.2) that lead to its current representative, the Web (2.2.3). Similar we portray the evolution of educational software (2.3.1) to learning management systems (2.3.2), where we focus on content management (2.3.3) and learning technology standards (2.3.4). We describe the vision and reality of repositories for interactive learning objects (2.4.1), and consider matters of software architecture allowing for object reuse, that is, software components (2.4.2), and within-component adaptability (2.4.3).

The second part presents the multi-level, component-based architecture of our “**GRIS/ILO Interactive Learning Objects**” (3), an adequate Web framework (4), and case studies (5).

We combine major concepts of interactivity into “MVC Interactivity” (3.1) to reformulate terminology in terms of Computer Graphics principles. Next, we propose and realize an “ORC-SG” architectural design pattern (3.2.1) that encapsulates matters of learning object state, appearance, functionality, and graphical visualization/interaction into reusable software components, respectively into objects, renderers, constraints (3.2.2), scene graph nodes, and user interface components (3.2.3). Our implementation provides a toolkit of basic components for containers, data structures, 2D/3D geometry, images/video, and physical quantities (3.2.4). A scripting architecture (3.3.1), generalized to a network model (3.3.2), renders software components adaptable and interoperable. We discuss our “Scripting Database” approach (3.3.3) to organize interactive learning objects in digital libraries, and illustrate some of our most urgent needs regarding future learning technology standards. Lastly, we suggest a visual scripting mechanism, “Drag & Drop Scripting” (3.3.4), which communicates learning object state and functionality between other hypermedia objects, or native applications.

Our **Web framework** consists in turn of a “Layered Database Model” (4.1.1), a template-driven courseware generator assuring consistent linking and easy maintenance (4.1.2), an offline tool for managing content, structure, and design (4.1.3), and online wizards (4.2.1) providing community support for learners (4.2.2) and authors (4.2.3). Community members may not only discuss, annotate, rate, and modify text and illustrations, but also interactive learning objects, software components, and scripts.

We give three case studies. While an “Electronic Webmaster” illustrates capabilities of our Web framework (5.1), two educational applications of software components contrast model-based visual programming with low-level component programming (5.2). A last showcase demonstrates fine-grained hypermedia interlinking, visual scripting, and community support (5.3) in an interactive Web-based courseware.

Our initial goal with this thesis is to provide design guidelines for developing reusable highly interactive learning objects. We do not consider open questions concerning intellectual property, billing, and quality assurance. The presented component-based technology offers graduation and interoperability in a Java-enabled Web environment. Similar, our digital library vision contains not only large-scale entities, but also multi-granular software components and sub-component pieces (scripts). We develop methodologies for authoring, customization, and personalization of interactive learning objects; the tools, however, remain prototypes. Lastly, although we feel our work is interdisciplinary, our proof-of-concepts target the SMET (Science, Mathematics, Engineering, and Technology) domain, and do not deny a Computer Graphics origin.

We do not question the **added value** of interactivity and hypermedia in education. Today, interactivity in SMET education is generally regarded as to be crucial. However, we agree in that it is mostly applied inappropriately. Interactivity and hypermedia do not make learning fun and teaching easy. Yet, they are able to represent complex models and processes, and relationships between objects or alternative views. We witness them as integral part of our education, and wish to leverage the use of highly interactive learning objects in Web-based teaching. Specific learning theories, didactics, and usability have been respected, but are mentioned rather implicitly, and only as needed.

Finally note that, in spite of the potential of adaptable components for realizing adaptive systems – which Kay promoted in his fifth argument (1.2) –, we rather stick to *Ben Shneiderman’s* preference for high-level interactivity [Shneiderman97]. We believe that future learning technology standards and software agents will provide an adequate base for **adaptive educational systems** [Brusilovsky96, Brusilovsky98], but current technology is still in its infancy.





## 2 Basics

2.1 Interactivity .....	18
2.1.1 GUI Characteristics .....	18
2.1.2 Perception and Cognition.....	20
2.1.3 A Qualitative Framework .....	22
2.2 Hypermedia.....	24
2.2.1 Origins.....	24
2.2.2 Design Principles.....	25
2.2.3 The Web .....	29
2.3 Web-Based Courseware.....	31
2.3.1 Educational Software .....	32
2.3.2 Learning Management Systems.....	36
2.3.3 Content Management.....	38
2.3.4 Learning Technology Standards.....	42
2.4 Interactive Learning Objects .....	45
2.4.1 Repositories.....	45
2.4.2 Software Components .....	52
2.4.3 Scripting.....	54
2.5 Conclusion.....	57

## 2.1 Interactivity

*“[T]he point is not: interaction yes or no. The point is: more or less. All the named characteristics of interactivity are gradients” [Jaspers91]*

An interaction describes an action (Latin: *agere*) between (Latin: *inter*) two or more participants; technology in human-computer interaction in turn provides a certain degree of **interactivity**. Members of the diversified educational community differ about the denotation of interactivity. Although interactivity has become an often-cited panacea for education [Aldrich98] – even more when combined with hypermedia – it is still trivialized to menu selection, clickable objects, or linear sequencing [Sims95].

In the following, we sharpen the term interactivity as it relates to Web-based teaching and emphasize its relevance for learning. We identify three approaches: the developer's view of interactivity as graphical user interface (GUI) characteristics (2.1.1), the educator's view of interaction between internal and external knowledge representation (2.1.2), and the communication theorist's view providing a qualitative framework (2.1.3). Later, we reformulate the results in terms of Computer Graphics principles (3.1) and illustrate the impact of a consequent implementation of a great interactivity with components of our own courses (3.2).

### 2.1.1 GUI Characteristics

Nowadays, we associate interactivity with multimedia. Ambron and Hooper [Ambron88] define multimedia as product of media (text, audio, visuals), technology (computers), and products (education, games, kiosk) – multimedia is not inherently interactive. However, the potential for interactivity might be multimedia's best distinguishing feature [Borsook91]; or, put another way, if any differences between different media can be found, it might be the interaction factor [Schulmeister97].

The relationship between humans and technology is well-investigated in the field of *Human-Computer Interaction* (HCI), which deals with the design, evaluation, and implementation of interactive computing systems, and with related human factors [Hewett92, Myers98]. In HCI terminology, an **interaction** is formed by a user action using a range of input devices (keyboard, mouse, touch screen, etc.) and resulting in some form of visual or audio output (text, graphics, etc.). As we focus on Web-based teaching, we restrict ourselves to input devices mouse and keyboard, and to limited Web browser GUI. Note that many of the HCI pioneers also broke new ground for hypermedia. Exactly 40 years ago, *Ivan Sutherland* presented the first interactive computer graphics system (*Sketchpad*); soon later, he joined the Department of Defense's Advanced Research Projects Center (ARPA), birthplace of the Internet. Think of *Douglas Engelbart's* NLS system – he casually invented the mouse and the first hypertext system – and his notions of connectivity and multiple views of information. *Ben Shneiderman*, who coined the term “direct manipulation”, developed *HyperTies*, the first hypertext system presenting illuminated, selectable

links. We will review the common history of HCI and hypermedia in section 2.2.

Today, interaction styles range from input by command line, menu, multiple choice, forms, spreadsheets, and natural language, to direct manipulation. Following *Ben Shneiderman*, **direct manipulation** [Shneiderman82, Shneiderman97] comprises (1) continuous representation of the objects and actions of interest, (2) physical actions or button presses instead of complex syntax, and (3) rapid incremental reversible operations whose effect on the object of interest is immediately visible. Direct manipulation lowers initial hurdles for novices as well as it enables experts to work more efficiently. Users get immediate feedback and gain confidence and mastery as they initiate and control actions and may predict system responses. The most prominent representative of direct manipulation, Drag & Drop (DnD, a shortcut for Copy & Paste) is supported by all major platforms (OLE/Win32 DnD, CDE/Motif dynamic protocol, MacOS, OS/2, and JavaOS/Java).

Facing the request for a better adaptability (and extensionality), we advance from the DnD gesture layer to the **programming** layer. *Alan Kay* envisions the computer as a personal, dynamic medium (1.2). The benefits of computer technology for facilitating learning are, in his words [Kay91], at first a "great interactivity", next, the hypermedia aspect of integrating all multimedia and representing information alternatively, and, last but not least, the capability of expressing and simulating dynamic models of ideas. Kay emphasized the need for a simplified programming framework; even children should be able to manage programming tasks [Kay77]. His ideas led to *Smalltalk* (2.2.2), the first object-oriented programming system. One of its design principles is the use of **building blocks** [Ingalls81], nowadays called software components (2.4.2), which represent reusable parts of applications. Apple Macintosh was the first to promote its widget toolkit (collection of GUI components) to enforce a consistent interface [Myers98]. Today, the Java Swing package contains about 40 GUI components, including the whole range from buttons, menus, input fields, and lists, to more advanced components such as WYSIWYG (what you see is what you get) styled text or HTML editors.

Software components come with several kinds of interactivity. *Rod Sims* [Sims00] identifies levels of interactivity with respect to the learner's role. His taxonomy provides combinable, **interactive constructs** that "can be integrated to provide comprehensive and engaging instructional transactions" [Sims95]. The proposed levels consist of object activation, linear and hierarchical interactivity, support, update, construction, reflection, simulation, hyperlink, non-immersive contextual (microworld), and immersive virtual (virtual reality) interactivity (see Table 1). Most of them can be directly mapped to corresponding software components. However, learning objects are typically composed of several components, and therefore cannot be classified clearly within this framework. Sims further gives clues on how to extend such constructs to provide statements about didactics or quality [Sims00]. We will incorporate these aspects in the next sections.

Construct	Description
Object	Object activation (e.g. button clicks) followed by a system response.
Linear	Forward/backward movements through a predetermined linear sequence.
Hierarchical	Linear interactivity preceded by a selection (e.g. menu selection).
Support	Optional performance support (e.g. general or context-sensitive help).
Update	Analysis of a user action, and generation of a matching update/feedback.
Construct	Problem solving requires manipulating component objects.
Reflective	Non-intelligent feedback opposing user's response with correct answer.
Simulation	User control; individual selections determine a training sequence.
Hyperlinked	Browsing a knowledge base.
Microworld	Training tasks of the work experience in a virtual environment.
Virtual Reality	Moving and acting inside of a complete virtual world that responds.

**Table 1: Rod Sims' taxonomy with combinable, interactive constructs can be directly mapped to software components [Sims95].**

### 2.1.2 Perception and Cognition

Interactivity design concepts such as direct manipulation or the desktop metaphor model use familiar instances of everyday life to bridge the gap between abstraction and reality. The homoiconic Smalltalk programming language expresses any characteristics of a system, even the language itself, uniformly – for users, internal and external representations are essentially the same. However, authors such as Aldrich, Rogers, and Scaife [Aldrich98] argue that, besides understanding interactivity in terms of “physical activities at the interface” or supporting models of learning, we need to analyze the “cognitive interplay between internal and external representations that arise in the different settings”.

Interactive environments typically use internal and external representations in concert. An **interaction** occurs as a perceptual or cognitive process when users utilize, adapt, or construct an external representation in a given activity (see cognitivism, 2.3.1). Common interactions are searching, parsing, recognizing, abstracting, re-representing, remembering, or keeping track of different stages of a problem or activity.

Following Rogers and Scaife [Rogers98], an **external representation** comprises four cognitive properties: computational offloading (how much do different external representations reduce the amount of cognitive effort required to solve a problem?), re-representation (if they have the same abstract structure, do they make problem-solving easier or more difficult?), graphical constraining (are the applied graphical elements of a representation able to constrain the kinds of inferences that can be made about the underlying concept?), and temporal/spatial constraining (do different representations make relevant aspects of processes and events more salient when distributed over time and space?). Based on such a framework, they develop **guidelines** for audiences such as developers, educators, or parents (see Table 2). A guide consists of “a set of questions and dimensions that [users] can usefully employ when thinking about the added value of interactivity” [Aldrich98].

Interactions occur (1) from external to internal representation and (2) from internal to external representation. The first direction describes **integration** of information. Domain knowledge in SMET often requires formal representations of complex, often invisible, abstract concepts. Rogers and Scaife [Rogers98] ask, “what is the best way of structuring different media, such that they convey the appropriate kind, level, and abstraction of knowledge for a given domain?” Different kinds of media and interactivity used in parallel allow for a more effective way of understanding concepts. All representations should be dynamically interlinked to visualize the relationships between them. They note:

*“[A] central question is: how can we determine the most effective way of displaying and coordinating multiple representations at the interface whilst at the same time supporting the interactions and activities which the user should be able to control and do for themselves?”*

We should allow learners to modify (correct or incorrect) elements in any representation. Effects of modifying components in one representation should be displayed simultaneously in all other representations. (A major part of our work will deal with this task.) Varying the level of computational offloading (the effort it takes to solve a problem) might look like this: introduce an abstract concept by depicting a simple illustration/animation of a concrete instantiation, then switch to an interactive learning object, and finally to a hypertext.

Explicitness and visibility	How to direct learner’s attention to key components, e.g. visualize normally “hidden” processes?
Cognitive tracing	How to allow users to manipulate and annotate dynamic representations?
Ease of production	How easy is it for users to create external representations?
Combinability and modifiability	How to enable the system and the users to combine different kinds of representations?

**Table 2: Rogers and Scaife develop design guidelines for interactive learning objects. Guides assist developers, educators, or parents with a set of questions and dimensions. [Aldrich98]**

The inverse cognitive process, **construction** of external representations, refers to learning methods such as highlighting text, making marginal notes, or sketching text-based ideas graphically. They ask, “to what extent can they be supported, simulated or extended at the interface?” Having a better understanding of how to create content will help users to understand the system. Moreover, it will enable learners to develop their understanding of the content by making individual changes to it (2.3.1). We should require the learner to test out hypotheses in different contexts, run a simulation, or build a model that will help the learner in developing a better mental model of the function and structure of a system.

Now, how does this approach map to physical interactivity? Both consider level of user control, extent of annotating, amount of feedback, and complexity of domain knowledge. Rogers and Scaife [Rogers98] note that their design concepts could be concretized “in terms of design parameters such as the type of media, the kinds of navigation aids, and use of color”. We end up with well-known **usability** guidelines; however, we believe (1.3) that usability guidelines provide only subjective hints and the usefulness of a learning object is mainly determined by the developer’s or educator’s mastery. To cultivate such competence, the presented catalog of questions and dimensions appears to be more adequate than a mere list of bits and pieces.

### 2.1.3 A Qualitative Framework

Direct manipulation constructs not only objects, but also cognitive concepts, such as geometric models, or relations of objects and parameters. *Rolf Schulmeister* [Schulmeister97, p. 341] strictly separates technical aspects of interactivity from its symbolical meaning by depicting learning by direct manipulation (technical) as ‘learning by constructing’ (symbolical). In moving towards the communication theorist’s point of view, we describe the nature of an **interaction** by methodologies based on theories of learning (2.3.1). Common models of Rhodes and Azbell [Rhodes85] or Schwier and Misanchuk [Schwier93] for example embody ideas of behaviourism (reactive learning, e.g. drill & practice), constructivism (proactive learning, i.e. self-active), and cognitivism (mutual learning, e.g. adaptive systems). Interactions now represent a learning process occurring while modifying objects. Consequently, hyperlinked interactivity (2.1.1) symbolizes no more interaction, but mere navigation [Schulmeister03, p. 209].

Current efforts in creating **standards** for learning object metadata classify interactivity to enable educators searching/browsing for learning objects in digital libraries. The *LOM* (Learning Object Metadata, 2.3.4) specification for example denotes the interactivity level within an ordinal range from very low to very high, but, in its current version, does not assign any characteristics to these ranges. This indicates only subjective impressions and rules out any international understanding [Schulmeister03, p. 208]. Some developers will weight the frequency of interactions, some will consider the quality, and others the multimedia type. Therefore, Schulmeister [Schulmeister03, pp. 210] proposes a qualitative framework consisting of six degrees of interactivity that can be directly mapped to

0	Observation (no interactivity)	The user contemplates a multimedia object (e.g. image, video, sound, automated program) and performs necessary media actions (e.g. start, stop), or navigation.
1	Observe multiple representations (illustrative actions)	The user may choose from a set of options and contemplate temporal (slow motion, step-wise) or spatial (point of view) versions of multimedia objects. Examples: slide show, alternative data lists.
2	Modification of representation (motivating actions)	The user may vary the visualization, but not the content of a multimedia object (e.g. pan, zoom, rotate graphical scenes and objects).
3	Modification of content (interaction with cognitive concepts)	Content is no more pre-prepared, but generated as response to the user. The user may create different visualizations or visualize different relations by varying parameters. Mostly found in SMET domain, e.g. parameter manipulation in physical simulations.
4	Construct objects or models (microworld)	The user constructs new objects and designs underlying models or processes. Examples belong mostly to the SMET domain where objects and processes can be expressed adequately, e.g. dynamic geometry software like <i>Cinderella</i> [Kortenkamp99].
5	Feedback (intelligent analysis)	The user gets intelligent responses according to his actions. Schulmeister mentions again <i>Cinderella</i> , which applies an automated theorem checking engine.

**Table 3: Schulmeister provides a qualitative framework for classifying a learning object's interactivity in six ascending degrees. Each degree includes all aspects of the lower ones. [Schulmeister03]**

LOM levels (see Table 3, and 2.3.4). With ascending degree, the related theory of learning alters from behaviourism to instructionalism to constructivism. The 4<sup>th</sup> level corresponds to learning by discovering, the 5<sup>th</sup> level to learning by construction.

Schulmeister implicitly assumes that developers design these levels properly. Consistent with LOM, each level includes all aspects of the lower ones. In practice, such an ascending order rarely occurs; a learning object as we know it rather presents a mixture of Sims' ingredients (2.1.1). A learning object may include feedback but no multiple representations, or it may offer construction without permitting modification of the representation. Also, from our point of view, feedback should be adequate, but not mandatory intelligent (which would require user/task modeling). Nevertheless, Schulmeister's proposition provides a far more elaborate and useful taxonomy than current standardization efforts.

## 2.2 Hypermedia

*“I have always imagined the information space as something to which everyone has immediate and intuitive access, and not just to browse, but to create.” [BernersLee99, p. 157]*

In this chapter, we review the common origin of hypermedia and graphical user interfaces. We contrast their most prominent representatives – the Web and the Desktop – in order to reflect on characteristics and shortcomings of interactive Web-based courseware. In contrast to other retrospections [Conklin87, Andrews96, Nielsen95, MuellerProve02], we outline milestones of hypermedia with regard to interactivity. System relationships are sketched in [Johnson89].

Following *Keith Andrews* [Andrews96, p. 13], **hypermedia** generalizes hypertext to include other kinds of multimedia in addition to text. Hypertext in turn consists of nodes and node connections – hyperlinks. A hyperlink is made of a source anchor specifying the starting point in a document, and a destination anchor defining a second location. Users navigate from source to destination anchor by activating the hyperlink, which we call ‘browsing’. Linking resembles the ‘goto’ programming instruction, and implicates similar problems. At first, it accounts for the “lost in hyperspace” syndrome [Conklin87, Maurer96, chapter 8.1] that describes user disorientation during browsing due to missing navigational aids. The second major shortcoming is the “broken link” (dangling link [Andrews96, p. 26]), which typically emerges during authoring or server migration, when a link’s destination anchor is lost, but the source anchor remains. “Real hypermedia” [Andrews96, p. 14] is interactive, integrates interactive multimedia and provides means for collaboration and personalization. We will see that all hypermedia pioneers intended to create a highly interactive medium.

### 2.2.1 Origins

We start our short history of the Web in 1945, the year when *John von Neumann* established the base of computers by describing concepts of a stored program, and *Konrad Zuse* developed the first programming language. *Vannevar Bush* described a microfilm-based *Memex* system, “a sort of mechanized private file and library” [Bush45]. He envisioned the use of **hyperlinks** and trails to archive scientific writings, annotate, and associate segments of the knowledge base, and to keep track of related data. Trails anticipated future guided tours, i.e. pre-defined paths along a given chain of thought.

In 1962, *Joseph Licklider* became head of the US Department of Defense’s (DoD) Advanced Research Projects Agency (ARPA), which was to improve the military’s use of computers. He redirected funding from private sector to university research institutions, such as *Douglas Engelbart*’s proposed “augmentation laboratory” at Stanford Research Center. At the time, computers operated in batch mode. Licklider dreamed of interactive computing going beyond punch cards (“man-computer symbiosis”, [Licklider60]). His vision of an “intergalactic” **network** engaging users in browsing, retrieval, and creation of new knowledge, laid the foundation for



ARPANET, the first Internet. DoD and the National Science Foundation (NSF) further facilitated the early public growth of the Internet, for example by enforcing the TCP/IP Internet standard.

Licklider's successor was *Ivan Sutherland*. His 1963 Ph.D. thesis at MIT, "*Sketchpad: A Man-Machine Graphical Communications System*" [Sutherland63], presented the first interactive computer graphics. Using a light pen and a 40-button command box, Sketchpad users could create, directly manipulate, duplicate, and store engineering drawings on the display. Constraints such as orthogonal lines could be applied, and the drawing area could be zoomed and scrolled. In the 1960s, batch mode processing usually occupied the computer exclusively for hours; therefore, this novel form of interactivity was called "on line". Sutherland invented what we call **real-time interaction** with computers.

About the same time, Douglas Engelbart worked out his HCI vision of instant connection and communication. Having a strong sense for automation and re-use, he recognized the importance of building tools – to spend a lot of time and energy first on building tools, then on applications. Based on Bush's vision, his *NLS* (On Line System) was the first point-and-click **hypertext system**, for which his group invented the mouse. In 1968, Engelbart demonstrated interactive text editing and groupware facilities such as screen sharing among remote users [Engelbart68, vanDam87]. His notions of connectivity and of multiple views of information would carry forward until today, but remain largely unrealized [Meyrowitz89]. NLS became the second node on ARPANET, redefining completely the concept of "online".

Parallel to the work of Engelbart, *Ted Nelson* coined the term hypertext, defining it as "non-sequential writing". He proposed *Xanadu* [Nelson65, Nelson82] in the 1960s, a system incorporating a hyperlinked **repository** for the entire world's knowledge ever published. Xanadu would include transclusion, a kind of inclusion by reference allowing reuse of content in multiple contexts. Work on Xanadu continues until today [Nelson99], but only parts have ever reached the state of prototypes. Among Nelson's visions were stretch text that elastically expands and contracts in place, and hypergrams, a kind of interactive illustrations [vanDam87]. Xanadu would have solved the broken link problem, as it provides version management, and global, unique identifiers for hypertext nodes. Versions are not deleted, but preserved forever by the system.

In 1967, Nelson collaborated with *Andries van Dam* to build *HES* (Hypertext Editing System, [vanDam69, vanDam87]) at Brown University. The text-based HES supported arbitrary-length content, and content reuse. Nelson left the group soon after realizing that HES was instead turning into a (the first) word processing system.

### 2.2.2 Design Principles

Heavily inspired by Engelbart's "mother of all demos", van Dam in turn redesigned HES to *FRESS* (File Retrieval and Editing System, [vanDam87]). FRESS offered bi-directional links and webs representing collections of links. Links had types, which could be employed, for example,

to indicate information about the link target (e.g. by a pictogram) before users actually activated the link. Link anchors were not stored in content, but organized separately in a **link database**, thus avoiding broken links (the link source can now be notified that its destination has changed) as well as it allows for referencing content where links cannot be embedded (e.g. read-only content, or video). It is also a prerequisite for personalization, e.g. annotation [Andrews96, p. 26]. FRESS also presented the first-ever undo facility. Van Dam's subsequent system *ELS* (Electronic Document System, [vanDam87]) further included graphical documents. It introduced page thumbnails, graphical links, and a **context** timeline for visiting recent pages and linked neighbors ("because context is so important"). The content's level of detail could be varied and navigation adapted according to traversed keywords. An authoring tool enabled the user to create pages, chapters, links, and graphics.

Context is also the reason why *Alan Kay* developed overlapping windows. In 1969, he laid ground for *Smalltalk* and windows with his Ph.D. thesis "The Reactive Engine" [Kay69] at the University of Utah (Ivan Sutherland is one of the committee members). Design principles behind *Smalltalk* can be found in *Daniel Ingalls's* writing [Ingalls81]; they led to Model View Controller and software **components** (2.4.2). The *Smalltalk* architecture had its core based on object-oriented programming with a uniform message system, enabling the user to interact with any aspect of the system. Objects could be adapted on the system level and interlinked system-wide. Objects referred to each other and sent **messages** in order to change their internal state, which extended the hyperlink paradigm naturally [MuellerProve02, p. 21]. Evolution has turned into just the opposite. Instead of a uniform programming system, today we face insular, application-centered operation systems; at best, applications provide restricted and non-conform scripting or macro functionality (2.4.3). Today, Kay's idea of a common programming platform is represented best by *Java*. Kay and Ingalls continue their work on *Smalltalk* until today; the *Squeak* system (presented in Kay's keynote at ED-MEDIA 2002 [Barker02]), e.g. transforms the *Smalltalk* environment into a standard Web browser.

Soon after, Kay and others founded the Xerox Research Center in Palo Alto (*Xerox PARC*), established to explore the use of computers in office environments. PARC researchers, among them members of Engelbart's group, create the modern GUI. *Frank Halasz's* hypertext system *NoteCards* enabled users to build new applications on top of it, and made it easy to customize the browser. It brought with it about fifty node types such as text, video, animation, graphics, and actions [Halasz88, Conklin87]. *Charles Simonyi* and others developed *Bravo*, representing not only the first WYSIWYG word processing application, but actually the first version of Microsoft Word [Johnson89]. *David Canfield Smith* introduced icons and **visual programming** in his Ph.D. project *Pygmalion* [Smith77], making it possible to connect components by direct manipulation (2.4.2). *Alan Borning's ThingLab* [Borning81] added to *Smalltalk* the Sketchpad notions of **constraints** (2.4.2), specifying relations that must be maintained. *Larry Tesler* articulated the concept of **modelessness**

[Tesler81, Meyrowitz89], demanding that users should always be able to easily understand their state within the system. Finally, in 1981 the Xerox *Star* system changed opinions about the design of interactive systems [Johnson89] by introducing the **WIMP** (windows, icons, menus, and the pointer device) metaphor. Apple Macintosh (1984) and Microsoft Windows 3.0 (1990) brought these concepts to the masses.

The 1980s put forth an immense number of hypermedia systems. Again, members of van Dam's group, among them *Norman Meyrowitz* and *Nicole Yankelovich*, developed the most sophisticated one, *Intermedia* [Meyrowitz86]. Object-oriented and WIMP-designed, it aimed for integrating all application programs into a hypermedia system. Meyrowitz argues that the failure of hypertext to reach its full potential stems from the development of "insular, monolith packages that demand the user to disown his or her present computing environment to use the functions of hypertext and hypermedia" [Meyrowitz87]. In contrast, *Intermedia* allowed for **interoperability** across a range of software components by providing a common API for all participating applications to share linking information. Linking followed the Copy & Paste metaphor: users set hyperlinks by selecting "Start Link" in any document's region and "Complete Link" in another arbitrary location. Links became a seamless part of the GUI:

*"Linking functionality must be incorporated, as a fundamental advance in application integration, into the heart of the standard computing toolboxes [...] and application developers must be provided with the tools that enable applications to 'link up' in a standard manner. Only when the paradigm is positioned as an integrating factor for all third-party applications, and not as a special attribute of a limited few, will knowledge workers accept and integrate hypertext and hypermedia into their daily work process." [Meyrowitz87]*

*Intermedia* was limited to Unix A/UX and would never be used widespread. Meyrowitz continued to focus media technology for the Internet; he developed *Shockwave* and became today's president of Macromedia. He sees the hypermedia system as the desktop of tomorrow [Meyrowitz89] excelling in integration, aesthetics, perspective, access, service, community, and adaptation (see Table 4). In short, the system must provide underlying technology, not applications.

With respect to integration, the potential of a **container** component model is illustrated best by *Andrea diSessa's Boxer* [diSessa85, diSessa86b] system. *Boxer* represents any information entity as box; a box may contain other boxes or data such as text or graphics. In contrast to other hypermedia systems, hierarchy is expressed naturally by nesting lower-level nodes directly within their parents. We can Copy & Paste any box into any other *Boxer* workspace. Flipping a box (face-up, face-down) offers an alternate view or internal state; for instance, we may flip a graphics box to see the rendering program. Programs are boxes with input/output variables together with other boxes specifying behavior. The *Boxer* language is homoiconic (like *Smalltalk*), that is, we can access, modify, and execute any data – which is also its major drawback: we are restricted to the proprietary *Boxer* environment.

Theme	Tasks of the Desktop of Tomorrow
Integration	Ease of linking, associating, combining, and incorporating components. Technology therefore has to provide a link database, link modes such as hyperlinking, warm linking (sending data), and hot linking (synchronizing data), and a container component model to create composites.
Aesthetics	Provide a sophisticated look (visual) and feel (operational). Allow users to develop components conform to the highest graphical design ideals, and maintain consistency, reliability, familiarity, and direct-manipulability.
Perspective	Provide multiple views of information. Support e.g. tasks, or trails.
Access	Allow for exploring, browsing, retrieving, and storing data. Technology therefore has to provide a database with a lens-like user interface.
Service	Provide standard general-purpose tools (linking, reference, linguistic services such as dictionary, thesauri, spell-checker, etc., and online services).
Community	Support user groups in synchronous and asynchronous collaboration (access rights, annotations, conferencing, shared editing).
Adaptation	Adapt to the user and, inversely, enable users to customize the system to perform functions that were not included by the system designers. Technology must allow for <b>scripting</b> to manipulate functions, let the user interactively manipulate an application's internal components. Applications should be able to register objects and methods that can be queried or manipulated by scripting.

**Table 4: Meyrowitz's vision of a hypermedia system is that of a desktop of tomorrow providing linking technology, a familiar look & feel, multiple views of data, standard tools (calendar, spell-checker, etc.), community support, and scripting. [Meyrowitz89]**

According to *Jeremy Roschelle* (developer of the Boxer graphics engine, who continues the research on software components until today, see 2.4.1), hypermedia's public breakthrough came with Apple's *Hypercard* in 1987, bundled free with every Macintosh system [Roschelle98]. HyperCard dramatically increased the number of educators able to produce their own interactive courseware. Users could customize objects, sequence screens of information by a stack metaphor, and construct buttons triggering scripts. The HyperTalk scripting language supported dynamic linking by computing links on the fly.

Between 1988 and 1990, developers of major hypermedia systems (among them Engelbart, Halasz, and Meyrowitz) abstracted their principles into the *Dexter Hypertext Reference Model* [Halasz90] to provide standard hypermedia terminology. It divides system architecture into three layers:

the storage, the within-component, and the run-time layer. While the storage layer describes a database containing a network of nodes and links, the run-time layer covers presentation, user interaction, and dynamics. The **within-component layer**, which describes content or structure inside of nodes, is not elaborated. Instead, the Dexter model formalizes a mechanism for specifying and anchoring to the interior of hypermedia objects independent from its current type (text, graphics, animations, or dynamic programs). Specified anchors remain constantly referable; only their value might change to reflect internal modifications.

Based on the lessons learned from Intermedia, in 1990 *Hermann Maurer's* group at Graz University of Technology began developing the hypermedia system *Hyper-G* [Andrews96, Maurer96], known today as *Hyperwave*. The Hyperwave server is designed to handle large amounts of multimedia data, which can be physically spread over multiple servers. The system incorporates technology such as a link database for bidirectional linking and automatic link consistency, and searchable object metadata (arbitrary name/value attributes). Underlying design guidelines are compiled in the *MANKIND* project [Maurer97]. The system fulfills Meyrowitz's demands for community support, as it provides access rights, annotations, shared editing, and a network protocol supporting synchronous communication and user sessions.

*“To ensure the success of a hypermedia system, it must allow users also to act as authors, allow them to change the database, create new entries for themselves or other users, create a personal view of the database as they need it, and, above all, allow the system to be used also for communication and cooperation.” [Andrews94]*

Hypermedia documents are stored in an object-oriented database using a set-based data model (clusters, 2.3.3). References to clusters have unique names (UNL); similar to Xanadu and the Dexter model, this avoids broken links.

### 2.2.3 The Web

In 1991, *Tim Berners-Lee* demonstrated the *World Wide Web* (WWW, W3) at CERN, inventing the Web's core independently from Nelson's and Maurer's work. An URL (Uniform Resource Locator) mechanism incorporates other Internet protocols such as FTP, Gopher, or Newsgroups; by this, “the Web' has embraced and become almost synonymous with ‘the Net’” [Andrews96, p. 10]. The Web uses *HTTP* (Hypertext Transfer Protocol based on TCP/IP) to transfer network data and *HTML* (Hypertext Markup Language) to separate content from design and structure. Two years later, NCSA (National Center for Supercomputer Applications at the University of Illinois) presented the graphical *Mosaic* browser rendering the World Wide Web the most popular web. From that time on, the **browser** client has driven Web technology.

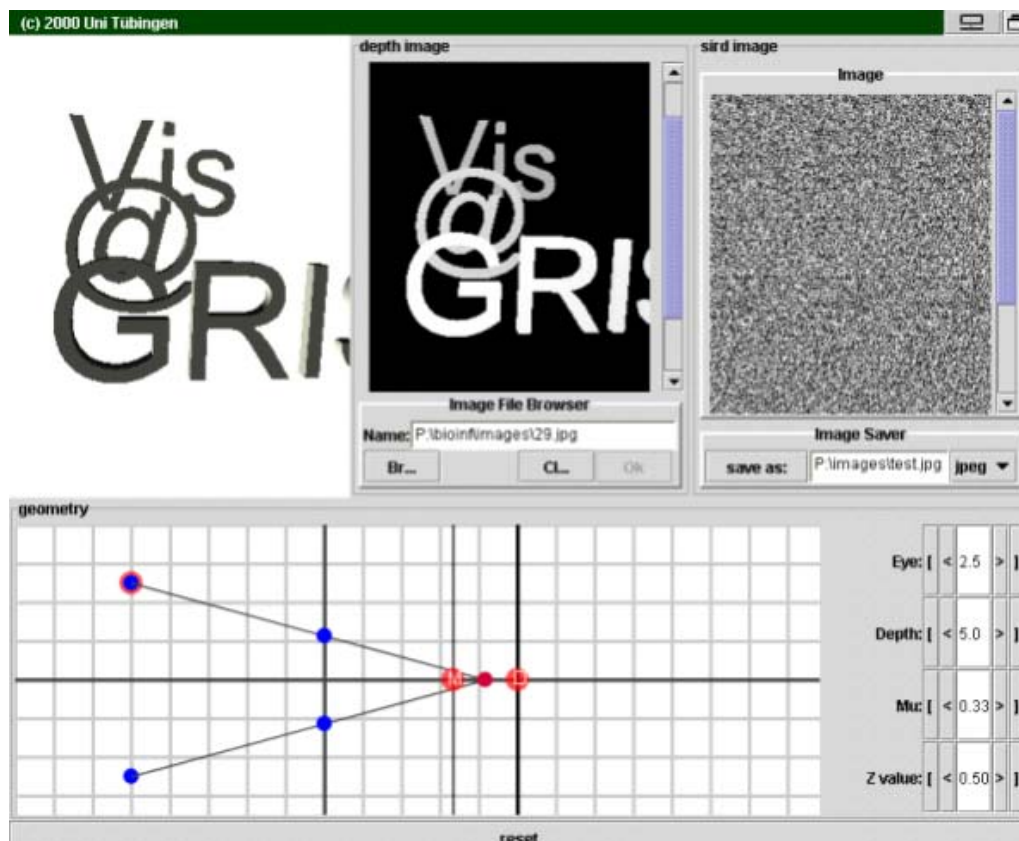
The more mature Hyperwave system could not turn around the Web's steady growth in popularity; in particular, its browsers (Harmony, Amadeus) could not keep up with standard Web browsers in terms of functionality. (Later, Hyperwave compensated this drawback by a Web gateway [HIM99] transforming protocols and delivering HTML to ordinary Web browsers). The limitations of the Web would lead to many

ambiguities. An URL contains the physical document location, so it may produce broken links. HTTP is stateless; in contrast to Hyperwave's protocol, it supports neither subsequent connections, nor user sessions. Lastly, with Mosaic many details of Berners-Lee initial proposal [BernersLee89] – such as typed links, multiple linking options (embed, jump to, show in separate window, etc.), scripting, and full integration of browsing and editing – are lost.

In 1994, ex-members of NCSA developed *Netscape Navigator* to seize Mosaic's commercial role. Microsoft released Windows 95 with built-in connectivity to their Microsoft Network (MSN) hoping it will replace the Web. Yet, they soon turned to a Web-based strategy, making MSN a Web site and developing *Internet Explorer*. This was the beginning of the browser war. Microsoft integrated the browser into its operating systems; Netscape in turn created the open source *Mozilla* browser. More than 100 browsers evolved (see e.g. the browser archive at <http://browsers.evolt.org>). Only a few could withstand Microsoft and Netscape's market role, e.g. *Opera*, which started as a research project run by Norway's telecommunication company Telenor, and would consistently manage to gain innovative features.

Also in 1994, Berners-Lee founded the World Wide Web Consortium (W3C), intended to lead the Web to its full potential by developing common protocols ensuring interoperability. Currently, W3C recommends 53 specifications [W3C03] targeting for example Web publishing (*HTML*, *XML*), linking (XLink, XPointer), 2D vector graphics (*SVG*), or multimedia synchronization (*SMIL*). The open source W3C *Amaya* browser seamlessly integrates editing and remote access, and serves as framework for other W3C technologies. Likewise, **standards** initiatives of the Internet Engineering Task Force (IETF) address Web issues such as the HTTP extension *WebDAV* (Distributed Authoring and Versioning, [Whitehead98]).

Despite of Kay's *Dynabook* vision, the Web still provides little interactivity, i.e. opportunities for learners to compose or construct their own ideas [Roschelle98]. Web browsers render content “read-only” and bring with them a “Web mode” incompatible to the Desktop (hyperlink paradigm vs. direct manipulation and WIMP). Missing dynamics features are compensated for either by server extensions (CGI – Common Gateway Interface), or, client-side, by **browser plug-ins** and scripting (DHTML). VRML [Bell95] introduced interactive simulations (virtual worlds) in 1995. At about the same time, Macromedia presented the *Shockwave* and *Flash* plug-ins for (pixel-oriented, respectively vector-oriented) interactive Web graphics; both are very popular today. Founded in 1992 to create interactive multimedia CD-ROMs, Macromedia provided authoring tools such as Authorware, Director, and Dreamweaver for creating Web and Flash/Shockwave content (with extensions for learning content). Flash is also integrated into two other widespread browser plug-ins for multimedia content, Apple *Quicktime* and RealNetworks *RealPlayer*. While focusing on audio/video, Quicktime further supports sprite animations and mouse interactions.



**Figure 2:** Java 3D provides access to low-level structures. Here, we access and visualize a 3D scene’s depth buffer (top left) as gray-valued image (top center) and feed it to a stereogram image filter (SIRDS, top right). Users may directly manipulate filter parameters (bottom line).

In 1996, the HTML 3.2 specification [W3C03] finally included Sun’s *Java* programming language. Initially part of browser functionality (Java 1.0/1.1), it has now become a standard browser plug-in (Java 2). The Java core is enriched with optional packages such as Java Advanced Imaging (JAI), Java Media Framework (JMF), Java 3D, and others [Sun03]. While providing high-level design concepts, they delegate rendering to low-level (native) libraries and hardware, e.g. Java 3D relies on OpenGL and Direct3D. Note that Java still provides access to low-level structures by encapsulating them into objects (see Figure 2).

### 2.3 Web-Based Courseware

*“But the problems that remain [...] are how to make sufficient room for two fundamental elements of scientific education in the training of the young. These elements are: (a) the genuine “activity” of the students, who will be required to reconstruct, or in part rediscover, the facts to be learned; and (b) above all, individual experience in experimentation and related methods.” [Piaget73, p. 34]*

Having reviewed the evolution of the Web and the Desktop as today’s representatives of hypermedia and graphical user interfaces, we now consider their use for education. We survey the classes of educational software and respective learning theories (2.3.1), and then present the

concept of a learning management system (2.3.2), which allows us to specify required properties, and compare our approach of Web-based courseware with others. Principles of content management such as a set-based data model, template-driven generators, and online wizards offering community support will be discussed in more detail (2.3.3). We close with a brief discussion of learning technology standards (2.3.4) that promise to provide adequate object metadata and interoperability.

### 2.3.1 Educational Software

The idea to support the learning process by the use of computers (computer-based teaching, CBT) and the Web (WBT) has lead to many variants and acronyms [Schulmeister97, p. 93, Blumstengel99, Maurer01b] such as CBE, CBI, CAT, CAI, CAL, CBL, CML, CMT, ICAI, or ITS. They accentuate respectively the base (B) of a computer (C) or an intelligent (I), interactive (I) system (S) to aid (A), assist (A), manage (M), or support (S) education (E), instruction (I), learning (L), teaching (T), training (T), or tutoring (T).

*Peter Baumgartner* [Baumgartner92] notes that any educational software is based on a **learning theory**, whether the authors aimed for it, or not. The “Theory into Practice” database [Kearsley03] contains more than 50 theories relevant to teaching and learning. Romiszowski [Romiszowski86] for example differentiates philosophical positions and their emphasis on useful content (Humanist), outcomes (Behaviorist), process (Cognitivist and Developmental), or system (Cybernetic). We restrict ourselves to the classic triad behaviourism, cognitivism, and constructivism (see Table 5, and [Schulmeister97, Blumstengel99, Sims00]).

<b>Learning Theory</b>	Behaviorism	Cognitivism	Constructivism
<b>Mind is...</b>	passive	interactive	autarkic
<b>Knowledge is...</b>	stored	processed	constructed
<b>We learn...</b>	“what”, facts	“how”, understanding	“how we come to know”, discover solutions
<b>Strategy</b>	teach, instruct	observe, help	cooperate, do not teach
<b>Interactivity</b>	hard-wired	dynamic	autonomous
<b>Educational Software</b>	tutoring system, drill & practice	adaptive system, intelligent tutoring system	microworld, simulation, hypermedia

**Table 5: Learning theories with underlying assumptions, paradigms, and related classes of educational software. Adapted from [Baumgartner92, Blumstengel99].**



Let us go back to the first attempts of CBT in the 1950s, the time when **behaviorism** had significant impact on education. A behaviorist concentrates on observable, evaluable behavior, and avoids referring to the learner's internal states such as feelings or cognitive processes [Kearsley03]. *Edward Thorndike* [Thorndike22] characterized learning as the result of associations forming between stimuli and responses. Behaviorism forms the base of today's classroom setting. Further, as a single teacher cannot appropriately assist 30 students simultaneously, *Burrhus Frederic Skinner* [Skinner54] suggested a teaching machine for individual use that could present and reinforce information, and choose the level of difficulty according to the learner's performance. Due to technological constraints, Skinner's theory of programmed instruction produced only gap-filling tests [Schulmeister97, p. 93] or linear "page-turning" programs. Behaviorism became associated with tutoring systems, trial & error, and drill & practice; typically, interactivity is hard-wired [Baumgartner92, Blumstengel99].

In the 1960s, *Jerome Bruner* [Bruner66] focused on mental processes and put down the theoretical foundation for **cognitivism**, which deals with the question of how we achieve, structure, conceptualize, and transfer information. Learning occurs by interactions between internal and external models (2.1.2). At first, efforts to optimize these interactions led to instructional design models. *David Merrill* for example developed the component display theory specifying ingredients necessary for efficient learning for a given objective and learner: rules, examples, recall, and exercise with feedback. He implemented his theory in the course structure of his *TICCIT* project (Time-shared Interactive Computer Controlled Information Television, [Merrill80, Schulmeister97, p. 98]) at the University of Texas and Brigham Young University. Many future authoring tools for instructional design would follow his principles. Adaptive systems and intelligent tutoring systems continue the search for a representation producing the most effective learning experience for a given individual and subject. (In some ways, reusable learning objects have the same intention.) *TICCIT* received funding of the *DoD* to determine whether CBT may reduce cost and time for training. The same initiative supported *Donald Bitzer's* group at the University of Illinois in developing the more technical *PLATO* (Programmed Logic for Automated Teaching Operation, [Bitzer70, Schulmeister97, p. 98]). *PLATO* introduced networking to CBT by offering a central computer with courseware and permitting to use it from terminals, communicate and include statistics and feedback [Maurer02]. Again, this corresponds with a principle of Bruner, namely the ability to verbalize to one and others (see community support, 2.3.3).

Bruner’s principle of discovery learning [Bruner61] – that we gain knowledge most effectively by personal discovery – leads to **constructivism**. He sees knowledge represented enactively (tactile experience through manipulating objects), iconically (visual stimulation through comparing and contrasting), and symbolically (abstract reasoning). While *Jean Piaget* [Piaget70] relates these modes to periods of childhood development, Bruner treats them as present and accessible anytime (with one of them dominant in each period). Piaget argues that we are born with a tendency to organize our thinking processes, and mental models consist of basic building blocks (schemes) and larger structures.

The work of Bruner and Piaget heavily influenced Xerox PARC researchers such as *Alan Kay* (see Table 6). Kay directed his Learning Research Group at PARC to develop a user interface that should explicitly address all three modes of understanding and manipulating the world around us (see direct manipulation and WIMP design, 2.2), an idea expressed best with their slogan “doing (mouse) with images (windows, icons), makes symbols (programming, Smalltalk)” [Kay93]. Piaget also collaborated with *Seymour Papert* at MIT, who developed *Logo* [Papert80] as a programming language for children in the late 1960s. Initially a mechanical robot connected to the computer, the turtle became a virtual plotter for creating vector graphics. In the 1990s, Logo was expanded to LEGO/Logo [Resnick91] linking Logo with the LEGO construction kit. The commercial *LEGO Mindstorms* products included new types of LEGO blocks such as lights, motors, and sensors for building machines, and new types of Logo building blocks (that is, software components) for building programs. The argumentation takes the line of constructivism:

*“In our experience, design activities have the greatest educational value when students are given the freedom to create things that are meaningful to themselves (or others around them). In such situations, students approach their work with a sense of caring and interest that is missing in most school activities. As a result, students are more likely to explore, and to make deep “connections” with the mathematical and scientific concepts that underlie the activities.” [Resnick91]*

DiSessa [diSessa86a] describes science learning as a re-experiencing process. Instead of learning a new concept through learning definitions, we must experience and re-experience the concept in different contexts, and gradually reorganize our intuitions into more complete models. Whereas a cognitivist regards learning as cognitive process occurring between external (the educator’s) and internal (the learner’s) model, a constructivist understands learning as the result of internal cognitive

<b>Bruner</b>	enactive	iconic	symbolic
<b>Piaget</b>	sensorimotor	concrete	formal
<b>Kay</b>	doing	with images	makes symbols
<b>GUI</b>	mouse	icons/windows	programming

**Table 6: Bruner and Piaget’s learning theories lead the creators of the first graphical user interface to a WIMP design (windows, icons, menus, and mouse pointer). [Kay93]**

processes – knowledge is constructed [Blumstengel99]. Following this principle of “learning without being taught”, Papert introduced the concept of microworlds [Papert80]. A microworld such as the Logo world of turtle-geometry represents an explorative learning environment that does not formulate concepts to be learned explicitly. Authoring tools offering visual programming implicitly establish associations between software components and provide the ability to manipulate the structure and behavior of microworlds at a high conceptual level [Birbilis00]. Discovery learning further provided a solid base for simulations, which expose the underlying dynamic model, and for educational hypermedia, which supports self-controlled exploration per se [Blumstengel99].

Following van Dam [vanDam87], Nelson was the first one to point out the importance of **hypermedia** in teaching. In the 1970s, van Dam applied *FRESS* (2.2.2) in schools. Students of an English poetry course had to analyze and critique a poem; the context included word glosses, references to other poems, and some professional analyses. In a second phase, they reviewed other students’ writings together with the teacher’s comments, and, in turn, reworked their analysis accordingly.

*“Electronic graffiti, as I thought of them. The reason I encouraged such annotations was that I remembered that when I was in college with Ted, I would always grab the dirtiest copy of a book in the library, rather than the cleanest one, because the dirtiest ones had the most marginalia, which I found very helpful.” [vanDam87]*

Consequently, the subsequent *Intermedia* system (2.2.2) was meant to prototype a future hypermedia system providing “a user-level framework for creating exploratory contexts of educational and research materials for students and faculty” [Meyrowitz89]. Apple’s *Hypercard* enabled educators to create dynamic educational hypermedia. Maurer [Maurer02] reflects on the era of authoring systems and underlying technology (e.g. interactive videodisks) before the advent of the Web in the 1990s. During the time the Web appeared, advanced authoring facilities were competing with the continuous discussion of learning models. This was not only a battle between authoring systems and browsers, but also between cognitivism and constructivism. The importance of communication and collaborative work grew steadily, as did the spread of large educational environments. The educational community started standardizing protocols and metadata. In the new millennium, the trend moved towards learning technology standards to provide a common platform for future educational software. Maurer states:

*“To use electronic educational material on a large scale it is necessary to be able to locate and re-use material created elsewhere. It is with satisfaction that we can observe a continued harmonization between North-American and European efforts in this important area. It is also gratifying to see that the necessity for communication, collaboration, and large digital libraries accessible via the Web is now universally accepted as basis for viable learning systems. [...] It has also become clear that a substantial learning system needs a huge array of administrative features and that the authoring of material is just a small part of what is necessary to make e-Learning work.” [Maurer02]*

### 2.3.2 Learning Management Systems

Imagine an ideal piece of educational software. What requirements should it meet? Educational software today is Web-based software, reflected, e.g., in SCORM's (Sharable Content Object Reference Model, 2.3.4) assumption of a Web-based infrastructure [ADL01a, p. 30]. Therefore, it first has to compensate for the limits of the Web and the Desktop (2.2). Basing a hypermedia system on the solid foundation of a learning theory – behaviorism, cognitivism, constructivism, or a combination of them (2.3.1) – entails further requirements concerning administration, learning, and authoring. The current trend of learning management systems (LMS) intends to meet both hypermedia and pedagogical needs. An LMS refers to a suite of functionalities designed to deliver, track, report on, and manage learning content, student progress and student interactions [ADL01a, p. 30, LSAL03, p. 5]. In contrast to proprietary educational software, LMS content represents reusable, interoperable entities. Learning technology standards (2.3.4) are supposed to provide the base required for an LMS to work.

According to Schulmeister, an **LMS architecture** [Schulmeister03] covers the domains administration, learning, and authoring (Table 7). A repository stores all data; it is realized as one or more databases accessed by public interfaces (API). Administrative functionality includes managing courses (i.e. course structuring, executing lecture/tutorial/homework), users with specific roles (student, educator, tutor, administrator, developer, etc.), and institutions (i.e. curricula, library, billing), as well as tracking user activities and allowing for self-tests and evaluation [Schulmeister03, p. 10]. Learning tasks cover course presentation and common tools for learning (calendar, mindmaps, etc.), communication (e.g., e-mail, chat, white board, discussion board), and personalization (of design, layout, and contents, e.g., annotations). Finally, authoring tasks include creating and modifying teaching material ranging from interface design to a particular learning object (e.g., text, slides, illustrations, animations, interactive objects) to exercises and tests (following a learning object, or spanning several ones). A must for any Non-English content is support of multiple languages, both in interface design and content.

Adminis- tration	Learning Environment	Authoring
user	courses	interface design
courses	communication	learning objects
institutions	tools	exercises
evaluation	personalization	tests
Interfaces – API		
<b>Repository – Database</b> user data, course data, learning objects, metadata		

**Table 7: Schulmeister's ideal LMS supports administration, learning, and authoring. Data is kept in a repository. [Schulmeister03]**

Present systems concentrate on parts of this ideal architecture only. For example, some systems provide only rudimentary administration functionality, or link to external learning objects instead of being based on an own repository. The smallest LMS consisting of a single course is called **courseware** – our own

implementation represents such a system. Variants such as a learning environment or a learning content management system rather accentuate learner activities, or content respectively. Schulmeister [Schulmeister03] and Baumgartner [Baumgartner02] evaluate more than 100 current LMSs and set up a feature list. Let us exemplarily characterize the senior of LMSs, *WebCT*.

*Example (WebCT): Murray Goldberg developed WebCT in 1997 at the University of British Columbia in Vancouver in order to facilitate course preparation and enrich students' learning experiences. Version 3.6 consists of a set of tools (CGI and JavaScript scripts, images, and Java applets) and uses HTML functionality without database functionality. The upcoming version will be based on Java and an Oracle database. Wizards guide course developers, administrators, learners, and graders in completing common tasks. Developers may arrange the course homepage (a separate HTML page), syllabus, content modules, calendar, discussion boards, mail, and chat. Tools allow for uploading files, arranging paths (linear trails), setting the course design (colors, page layout), and setting up user accounts. Authors may toggle between a presentation and a design mode; in the latter case, an integrated HTML editor simplifies the production of WYSIWYG content. Evaluation tools allow for tests and polls. Its interface supports multiple languages.*

Many of the described tasks meet more general requirements of an **hypermedia system**. Remember Meyrowitz's demands for a desktop of tomorrow (2.2.2): not applications, but the system must provide means for easy component composition and linking, as well as a uniform look & feel and alternatives in structure (e.g. trails) and content (e.g. multi-lingual, adaptable multimedia). Common services such as calendar, e-mail, annotations, chat, white board, discussion board, etc. must become system services.

*Example (Browser history): The navigation history of today's Web browsers represents a list of visited pages. An LMS has only limited access to its internal structure: in general, only relative movements, forwards or backwards, are possible (by scripting). Imagine the history would become a system service instead of a browser feature. History items (i.e., visited Web pages) would be stored in a standard Desktop folder. The LMS could sort, search, edit, group, share, archive, etc. the history with familiar Desktop facilities. Note that we can interpret trails similarly.*

*Example (Calendar): Each LMS provides its own, proprietary calendar tool. If it became a system service, we could reuse the calendar in any application, e.g. in the user's e-mail program or word processor, and any given data could be shared with others using standard Desktop network security and access rights. Microsoft's server-side scripting environment ASP.NET suggests a Web-based solution – it enables developers to include an interactive calendar component simply by using the statement `<asp:Calendar runat="server"/>`.*

Typically, an LMS lacks in the authoring domain. It integrates only rudimentary support for WYSIWYG hypertext editing (as in the case of

WebCT) and delegates all non-trivial authoring facilities to external **authoring tools**. Again, evolution takes the application-centered line (cp. with 2.2.2) instead of integrating authoring functionality into the system's core. Authors may create, e.g. hypertext with a WYSIWIG editor (e.g. Microsoft Frontpage, Macromedia Dreamweaver), code editor (e.g. Macromedia Homesite, Emacs), or their favorite word processor (e.g. LaTeX, Microsoft Word, Adobe FrameMaker) using the hypertext export facility [Klein98a, Wiest01]. Likewise, we find applications for text-based questions and tests (click2learn's Toolbook Instructor, Macromedia Authorware, Microsoft LRN) supporting the related IMS specification [Smythe02]. IMS covers only multiple-choice tests or drag & drop puzzles; other tests, such as true/false, multiple choice, drag & drop, match item, text entry, or hot spot (identification of a particular region of the screen), graphics (e.g. Corel Draw, Adobe Photoshop), animations and interactive objects with medium complexity (e.g. Macromedia Director, Flash, Shockwave, Microsoft Powerpoint), are created separately. In the case of highly interactive objects, not only content, but also the programming architecture is handcrafted. Integrated developing environments (IDE) allow for implementing, assembling, and testing software components and applications, which are, in the case of Java, available for free (e.g. Sun ONE Studio, Borland JBuilder, Eclipse).

In a word, we find authoring tools and programming architectures separated from LMS architecture. Resulting courseware shows up fundamental deficiencies in modifications and interlinking: learning objects, in particular interactive ones, become **black boxes** that can neither be modified sufficiently (e.g. choosing parameters or enhancing functionality) nor be interlinked properly with their context (e.g. synchronizing an object's state with a guided tour). All-in-one applications claiming to create unified, interlinked, and interactive courseware (e.g. Toolbook or Director) reach a proper interlinking by the use of proprietary Web formats – now the whole courseware appears as a read-only, autarkic black box (e.g. a Flash application with interlinked text, images, etc.).

### 2.3.3 Content Management

As a Web-based information and communication system, an LMS must naturally deal with the organization, production, delivery, and update of Web content, that is, with content management. Nowadays, the appropriate Web framework [Hanisch00b] consists of an underlying database, a template-driven generator, a standard Web browser, and online wizards (see Figure 3).

We obtain efficient functionality for data organization by **database** technology. Actions such as query, select, insert, or update can be expressed in the standardized *SQL* (Structured Query Language), which abstracts from data representation and describes operations in a declarative way (goal-oriented, with a syntax like “select [items] from [table] where [criteria]”). Hyperwave (2.2.2) and Home [Duval95] further employ a set-based data model providing “support for orthogonal yet closely coupled structuring, linking, and search facilities” [Andrews96, p. 30] as system service. Documents can be grouped into collections or

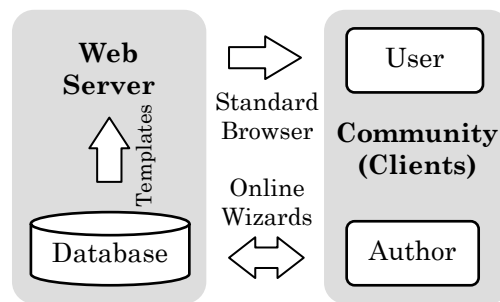
clusters to implement, for example, alternative media types and content, multilingual data, and versioning. They have a rich set of associated metadata useable in queries or data visualization.

*Example (Multilingual documents with clusters): While Web technology provides support for multiple languages by setting a browser variable (`HTTP_ACCEPT_LANGUAGE`) and appropriate server mappings for content*

*negotiation, this technique is rarely used. An approved method is letting the user choose the language on the entry page and then follow different paths through the server for different languages [Schmaranz96]. Consider the consequences in a courseware scenario, coming along with content in two or more languages, in multiple versions (short, medium, and long versions), and in alternative formats targeting screen (HTML), print (PDF), network bandwidth (ASF/RM/QT video streams), or plugins (SVG, MathML, etc.). Authors could hardly create Web pages consistently, or include functionality to change language settings at arbitrary locations. Clusters, on the other hand, allow for adjusting content on the fly: users specify preferences through metadata, e.g. `English/long/screen/58k/MathML`, and the matching Web page is generated system-side. Because of assembly works on sub-document layer, clusters potentiate content reuse – the database may e.g. store language-independent blocks of a page only once.*

Content management separates content, structure, and design – a necessity not only if we want to reuse objects in a different context, but also for pedagogical and content/form reasons. Maurer [Maurer97] notes that, based on the many lessons learnt, the system must include built-in guidance for courseware design (but not dictatorship), like e.g. the use of fonts, color, graphics, guided tours, or feedback. Usually, design **templates** steer courseware generation from the core data, resulting in a uniform look & feel. They provide orientational and navigational aids, as well as an automatic structuring and maintenance of information [Andrews94, p. 27]. The *Exploratives* project classifies common structural patterns as reusable hypertext building blocks [Spalter00] such as Locator (“you are here”), Overview (accentuated already by Shneiderman [Shneiderman89]), Lecture, Laboratory, etc.; it is not meant to be an exhaustive classification, but a basic support for authors to improve standard navigation behavior. Authors may modify courseware design globally simply by editing the corresponding template. Using different templates for the same content creates different views.

*Example (Link template) Remember that hyperlinks are part of the database (see Section 2.2.2). A link template may visualize the link type by a pictogram, validate the link target, and include further hints into*



**Figure 3:** LMS content management deals with the organization, production, delivery, and update of Web content.

*the main text, mouse-over text [Nielsen95], browser status bar, etc., in order to anticipate the target's type and content, e.g. a chapter title, file size, or illustrative thumbnail. Another link template may follow a link structure (forwards and backwards) and visualize a global or local graph. Electronic text books use many such templates, for example (1) the linear path to a chapter, (2) all subchapters of a given chapter, and (3) back-links to chapters that cite the current one.*

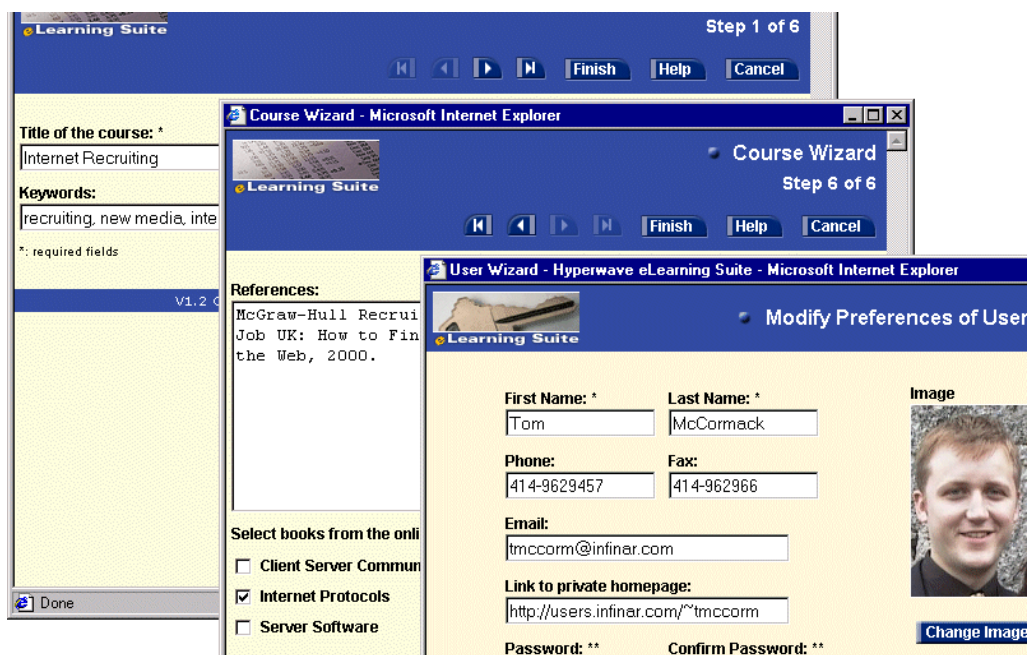
A template contains statements, loops, and variables that are filled up with appropriate data. It is represented textual or hard-coded, and carried out by an appropriate **generator**. A generator works either as an independent application (creating static Web pages frequently, e.g. once per day), or server-side (dynamic) as server add-on (e.g. CGI or Java Servlets) or server-side scripting (e.g. JSP, ASP, or PHP). While the latter ones are attractive for up-to-date information and personalization, they lack – compared to static generators – in speed, stability, and offline utilization (see Table 8). The major drawback of static generators, that changes are not visible instantly, can be compensated by content-specific server-add-ons (e.g. a discussion board servlet). Such hybrids balance static and dynamic pages according to the nature and frequency of updates. Note that generating static pages already is a form of caching.

Content management deals further with **community** support, that is with archiving, expanding, sharing and transferring a community's knowledge – the non-intelligent part of knowledge management [Maurer01a], in contrast to computerized or artificial knowledge. The LMS *Gentle* [Dietinger98] (based on Hyperwave, and recently renamed to Hyperwave eLearning Suite [HIM00]) for example collects all knowledge attached to a document (questions, answers, remarks, discussion boards, etc.) and

Frequently-Generated Static Pages	Dynamic Pages
<ul style="list-style-type: none"> <li>+ responsiveness &amp; scalability</li> <li>+ offline versions (e.g. CD-ROM)</li> <li>+ stand-alone (server/system independency)</li> <li>+ can use 3<sup>rd</sup> party search engines</li> <li>+ less issues integrating 3<sup>rd</sup> party tools</li> <li>- changes are not visible instantly</li> <li>- generation might be time-consuming (in large, dynamic Web sites)</li> <li>- server replication might be complex (e.g. in distributed environments)</li> </ul> <p>→ rather small, static Web sites</p>	<ul style="list-style-type: none"> <li>+ up-to-date information &amp; templates</li> <li>+ support changing contents (news, banners, advertisements, etc.)</li> <li>+ easier to do personalization</li> <li>- require know-how to avoid deficiencies in responsiveness &amp; scalability (e.g. caching, cache pre-loading)</li> <li>- extra work required to create offline versions</li> <li>- require specific server/system/tools</li> <li>- costs (usually per server per CPU)</li> </ul> <p>→ large, dynamic sites (e.g. kiosk, news)</p>

**Table 8: While dynamic Web pages are attractive for up-to-date information and personalization, static ones offer speed, stability, and offline utilization. A hybrid generator balances them according to the nature and frequency of updates.**





**Figure 4: Gentle wizards manage course structure, content, and users online [HIM00].**

includes it into the knowledge base. Maurer argues that, after a warm-up phase with human specialists, the system can generate potential answers for incoming requests automatically. Search is resolved bottom-up on sub-document level, from words to paragraphs, chapters, and documents.

As casual users will not accept database interfaces or simple input masks, an LMS must provide authors and learners with adequate input facilities. Online **wizards** allow distributed modification of the courseware on the fly [Helic00]; user input is versioned [Maurer96] and, after verification of the editorial staff, integrated automatically. A rating system assures quality by maintaining profiles for authors, learners, and content. Background on Web-based groupware functionality can be found in the domain of CSCW (Computer Supported Cooperative Work, [Bentley97, Dix96]).

Online wizards work task-dependently; *Gentle*, e.g. provides wizards to manage courseware structure, content, and users (see Figure 4, [Dietinger98, HIM00]); *WebCT*, on the other side, only provides basic, simple-structured interfaces (2.3.2). Typically, they revert to templates for structuring common, re-occurring workflows [Baumgartner02], and to define rules and constraints for user authorization, data input steps, preview, and database actions.

*Example (Java Struts)* The Struts framework [Apache00] for developing online wizards is based on Java Servlets and a modified Model View Controller design (2.4.2): while the View contains static “template” text and dynamic content based on the interpretation (at page request time) of special action tags, the Model contains the system’s internal state and the actions that we can take to change it. The Controller, finally, focuses on receiving HTTP requests from the client, decides what action is to be performed, and then delegates responsibility to the subsequent View.

### 2.3.4 Learning Technology Standards

Learning technology standards specify learning object metadata and interoperability. They aim at enabling educators to search/browse digital libraries, integrate the object of preference into their curriculum, and adapt it to their needs. Based on appropriate standards, an LMS could launch third-party learning objects, track them, and adapt them to the user.

So far, we have used the term learning object without explanation; let us now briefly clarify its meaning. According to the IEEE Learning Technology Standards Committee (LTSC, see below), any entity, digital or non-digital, which can be used, reused or referenced during technology-supported learning, represents a **learning object** – in short, any resource that we can reuse to support learning. Wiley [Wiley00, Chapter 1.1] surveys the many names of learning objects, among them knowledge objects (Merrill, 2.3.1), pedagogical documents (ARIADNE, [ARIADNE02]), sharable content object (SCORM, [ADL01a]), online learning materials (MERLOT [SmithGratto02]), and educational components (ESCOT, [Roschelle99]). Some authors create educational material but are not aware of it or do not care, e.g. in the realms of games or virtual reality. Terminology for interactive learning objects is tangled even more: variants range from interactive multimedia instruction [ADL01a], interactive illustrations [Beall96], explorable microworlds [Papert80, Birbilis00], exploratories (combining exploratorium and laboratory [Simpson99]), virtual experiments [Hanisch99], to games, manipulations, and simulations [MathForum03].

Wiley notes that the **building block** metaphor applied to learning objects (technically, 2.2.2, or pedagogically, 2.3.1), leads to wrong assumptions. In LEGO land, a LEGO block is compatible with any other LEGO block. Assembly is fun and simple; anyone can put them together. Six standard 2x4 blocks correspond with over 100 million combinations. Obviously, we prefer combinations with educational value. Learning objects in the SMET domain will mostly require theoretical background and training. A single statement is true: we need standards to ensure learning objects' interoperability.

The longing for interoperable hypermedia documents already showed in the work of Engelbart and Nelson, and produced standards such as TCP/IP, HTTP, and HTML/XML (2.2). Records of normative standards are kept by organizations such as the Institute of Electrical and Electronics Engineers (*IEEE*), the International Organization for Standardization (ISO), and the International Electrotechnical Commission (*IEC*). Learning technology standards are still evolving, and we act on **specifications** instead (see Table 9). *Erik Duval*, who is both technical editor for the standard on learning objects metadata (LOM) and president of the ARIADNE Foundation, outlined the current state of learning object metadata, participating organizations, and future roadmap in his workshop at ED-MEDIA 2002 [Duval02]. We may expect a formal ISO learning technology standard in a few years.

<i>ADL</i> (1997)	Advanced Distributed Learning Initiative. Started by the DoD and the White House Office of Science and Technology Policy, ADL collects and recommends best-practice specifications in the SCORM standard, and creates testbeds for learning objects and LMSs.
<i>AICC</i> (1998)	Aviation Industry Computer Based Training Committee. Provides interoperability guidelines for LMSs, especially to standardize training technologies for aviation industry.
<i>ARIADNE</i> (1995)	Alliance of Remote Instructional Authoring and Distribution Networks of Europe. Creates tools and methodologies for the share and reuse of learning objects (see KPS, 2.4.1). Includes a strong emphasis on respect for multiple (European) cultures and languages.
<i>IMS</i> (1997)	Instructional Management Systems Global Learning Consortium. Initially a project of the EDUCAUSE association, the scope of IMS specifications include Web-based course management systems, content metadata and packaging, and question & test.
<i>LTSC</i> (1998)	IEEE Learning Technology Standards Committee. The LTSC Learning Object Metadata (LOM) working group P1484 collaborated with ARIADNE and IMS to produce an educational metadata IEEE/LOM contains technical, didactical, and legal metadata, and is widely respected.
<i>PROMETHEUS</i> (1999)	PRomoting Multimedia Access to Education and Training in EUropean Society. An open European forum to improve the quality and use of learning technology, and to protect European multicultural values.

**Table 9: Several organizations collaborate to specify learning technology with regards to metadata, interoperability, and LMSs. Great interactivity is not yet considered.**

In their current state, learning technology standards do not support highly interactive learning objects properly. However, we believe that future specifications will include advanced issues of interactivity. We have therefore chosen to sketch the application of the Sharable Content Object Reference Model (*SCORM*, [ADL01a]) to interactive learning objects. Later on, in the context of our own projects, we will describe our most urgent needs (3.3.3).

The SCORM can be traced back, again, to interests of the Department of Defense (DoD), which spends more than \$17 billion annually for military schools that offer nearly 30,000 military training courses to almost 3 million military personnel and DoD civilians, many of them aimed maintaining readiness [GAO03]. Most courses occur at centralized training facilities; they span weeks or months. DoD plans to convert about 50 percent of these courses into online courses with estimated costs of \$10,000 per hour. (Note that less than five percent of DoD training programs

routinely uses interactive training technologies [ADL01a, p. 28] – a more frequent use would surely raise costs.) To avoid unnecessary expense, the DoD started the ADL initiative, which “envisions the creation of [...] repositories where learning objects may be accumulated and cataloged for broad distribution and use” [ADL01, p. 12]. ADL integrates four major areas into the SCORM: metadata, course-structure format, data model, and an application interface. It mandates how educational material must be organized, structured, and indexed so that different LMSs can cooperate in delivering a courseware to a learner. In SCORM terminology, course text, illustrations, tables, scripts, and highly interactive learning objects represent assets, arbitrary pieces of a sharable content object (SCO).

*“The content structure can represent a content aggregation ranging from very, very small learning resources – as simple as a few lines of [HTML] or a short media clip – to highly interactive learning resources that are tracked by an LMS. The Content Structure is neutral about the complexity of content, the number of hierarchical levels of a particular course (i.e., taxonomy) and the instructional methodology employed to design a course.” [ADL01b, p. 114]*

Let us investigate how the SCORM **metadata** (which, in turn, applies LOM) supports interactive learning objects. Metadata categories deal at first with general information and issues relating to lifecycle, rights, annotation, and classification. Assigning them is normally trouble-free. The same is true for technical metadata stating format (MIME types), size, URL location, and requirements (e.g., browsers, plug-ins). Regarding educational metadata, SCORM restricts vocabulary to a set of pre-defined values; only for specifying a learning resource type, SCORM suggests best practice vocabulary or user-defined terms. To specify an interactive learning object, we would combine some best-practice vocabulary for the learning resource type: exercise, simulation, questionnaire, diagram, figure, graph, index, slide, table, narrative text, exam, experiment, problem statement, and self assessment. Further details would be given as custom vocabulary, and as free text description.

SCORM differentiates between two types of interactivity type: learning-by-doing (active) and learning-by-reading (expositive). Simulations belong to the first type, as do questionnaires or exercises. Second type representatives are e.g. video clips, or hypertext. We can denote the interactivity level within a range from very low to very high, which suffice our needs – provided that SCORM will assign certain characteristics to these ranges, e.g. statements about physical interaction style (e.g. command line, menu, direct manipulation, 2.1.1) and cognitive activities (observation, modification of objects/model, feedback, 2.1.3). We also lack metadata for dynamic behavior (point & click vs. continuous interactions), bi-directional interactivity, and input devices (keyboard, mouse, touch screen, etc.).

The SCORM runtime environment [ADL01c] specifies **interoperability**, that is, how the LMS launches and tracks learning objects, and how objects exchange information. Current SCORM documentation considers learning objects as black boxes; in particular, it does not address intra-learning object branching or navigation within the learning object [LSAL03, p.39]. Objects may communicate with the LMS and store and retrieve string values (resolved through JavaScript calls); therefore, it

must support an appropriate interface. We can use this mechanism to launch an object in a specific state, or store its current state for later use. However, the model prohibits learning objects to set values of other ones (or ask the LMS to do so). If we want to interlink different learning objects, e.g. to synchronize their states, we would have to bundle them into a single learning object. Another motive for creating such bundles is SCORM's current launch model: it allows only one learning object to be active at a time – despite the fact (2.1.2) that learners usually work simultaneously with multiple objects, for example, with synchronized theory, exercises, and exploratory learning objects.

## 2.4 Interactive Learning Objects

In the discussion of interactivity in WBT, we portrayed the evolution of educational software to LMSs (2.3). Such systems obtain reusable educational modules (learning objects) from an integrated or external repository. In the following, we concentrate on a particular subset of learning objects, namely highly interactive ones. We describe the vision of digital libraries, and actual repositories holding interactive learning objects in the SMET domain, mainly in the field of Computer Graphics (2.4.1). Then, we consider – top-down – matters of software architecture allowing for object reuse. We provide an insight in software components (2.4.2) and adaptability (2.4.3), which refers to the within-component layer.

### 2.4.1 Repositories

The rising interest in repositories for educational material consolidates efforts of the diversified educational community. Developers, teachers, and designers have recognized the need for collaboration in order to create the best-possible learning objects, and, moreover, to preserve and reuse them. This is true in particular with regard to the production of interactive ones, which has proven difficult and extraordinarily time-consuming [Spalter03, Roschelle98]. Spalter and van Dam, for example, estimate “the cost of creating a single well-designed, highly graphical, and interactive online course in the commercial domain from several hundred thousands dollars to a million or more”. They note also that production by and for the educational community entails social issues such as limited funding or timescales that preclude the application of reusable modules.

The US National Science Foundation (*NSF*) has started a major research initiative on the design and creation of **digital libraries** [NSF03]. It supports the vision of a national SMET education digital library, which combines curriculum material developed in previously funded NSF projects, enhances current SMET education, and includes future material [Owen00]. As examples, we discuss results of the NSF-funded projects Exploratories, ESCOT, EOE, and MERLOT, and further include two European projects, ARIADNE and E-Slate, which are funded by European Union Research & Development or national projects.

A digital library is meant not only to provide Web-based housing of learning objects with browsing functionality – which is the task of a repository –, but also to offer advanced services (via well-defined protocols)

for structuring and accessing material, as well as for community tasks. Typically, a specialized staff ensures the library's integrity and continued existence over time [Owen00, Yaron01]. For practical reasons, content is limited to some specific domain. Here, we focus on research problems of digital libraries concerning matters of collecting, describing, finding, reusing, and adapting interactive learning objects. Problems with intellectual property issues, social issues, system specificity, and others can be found, e.g. in [Spalter03].

We have pointed out that learning technology standards provide the technological base for a digital library (2.3.4). However, currently they neither support matters of great interactivity, nor advanced interoperability. As an intermediate solution for accelerating production and developing adequate component technology, several projects therefore created **repositories** for interactive learning objects and software components (see Table 10). The comprehensive *Web/Comp* project [diSessa01] of Boxer-father *Andrea diSessa* (2.2.2) gives further insights into state-of-the-art repositories and component-based educational computing.

Projects developing interactive learning objects typically develop a Java component architecture, which – following Kay's line (2.2.3) – renders any object inspectable, modifiable, and extendable. Roschelle [Roschelle98] mentions that technically, the idea of a programming language as system architecture has proved problematic due to the difficulties in keeping in pace with rapidly advancing interactive technology. Consequently, Microsoft's C# language supports diversity in programming languages. A language-independent approach would also standardize repositories that, for now, collect interactive learning objects in various media types such as Flash, Shockwave, Director, Authorware, Toolbook, SVG, etc.

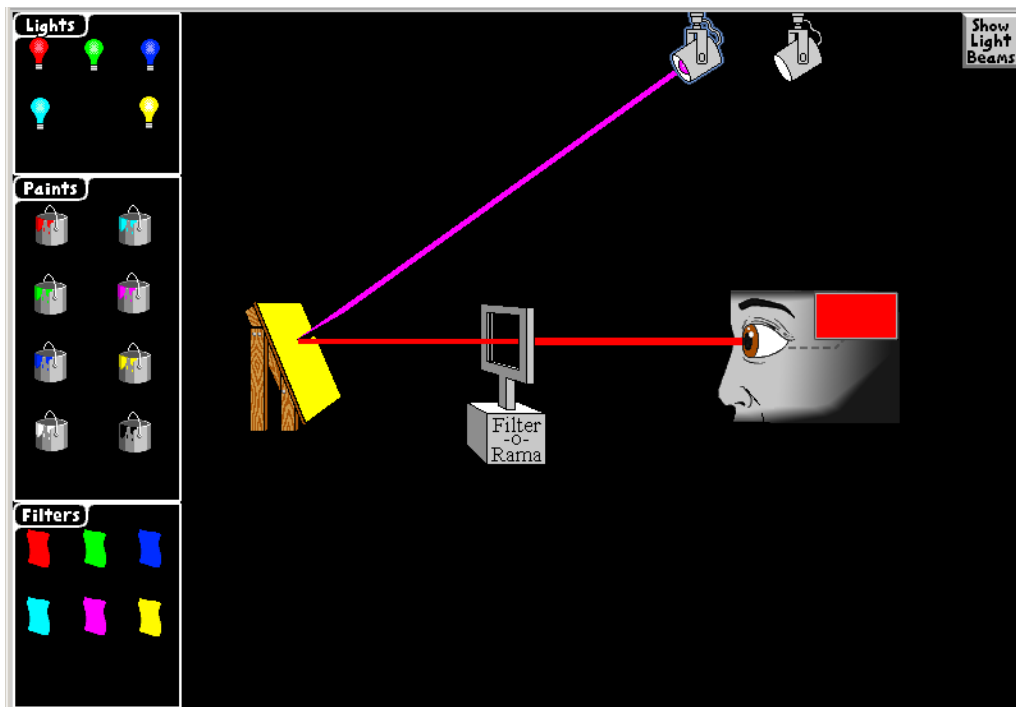
The *Exploratories* project [Beall96, Simpson99, Spalter03] publishes findings about strategies for creating and using educational Java applets. Following the line of other projects of *Andries van Dam's* research group (2.2), it represents Brown's most current contribution to Web-based teaching. Their **granularity** strategy [Laleuf01] decomposes a learning object into a graduated, multi-level component hierarchy from self-complete applications to components to sub-components, providing objects at all levels. All their applets use an "Exploratory" base component to support a consistent baseline of features (e.g. startup behavior, basic menus, basic container). In contrast to projects like E-Slate or ESCOT (see below), this approach enables programmers to access all aspects even of complex components. On the other hand, project members had to state that creating a complete collection for a single course leads to a huge effort that is hardly to manageable for a non-profit research group of ten people. Like similar projects, they report technical problems with browser compatibility (i.e. Java plug-in versions, JavaScript functionality, security settings) and system specificity (i.e. setup, user permissions, classpath, network issues). Some stated problems are typical in the university environment, such as student's lacking prerequisites in software design (i.e. object-oriented programming, component reusability).

Exploratories (1996)	The Brown University Computer Graphics Research Group's latest effort to leverage the computer's potential for use in education follows the line of Andries van Dam and others (see HES, FRESS, and Intermedia, 2.2.2). In developing interactive microworlds for teaching introductory Computer Graphics, the project explores the use of software components and strives for understanding underlying design patterns.
E-Slate (1993)	Manoulis Koutlis and Thanasis Hadzilakos at Computer Technology Institute, Greece, originally planned to create microworlds for learning Geography. Funded by national and European Community based Research & Development projects, it has become a long-term project going through many educational and end-users collaborations, as well as many technological re-designs.
ESCOT (1998-2001)	Educational Software Components of Tomorrow. Jeremy Roschelle, Chris DiGiano, Roy Pea, and Jim Kaput at SRI International's Center for Technology in Learning explored the use of Java-based component technology in math education. Focusing on reusability, they interconnected third-party JavaBeans components and substantiated the use of a scripting architecture.
EOE (1994)	Educational Object Economy. James Spohrer founded the first and largest repository of educational Java applets at Apple's Learning Communities Group in order to enable the formation of an educational developer community. EOE features 2,600 applets, with a fair number of broken links.
Merlot (1997)	Multimedia Educational Resource for Learning and Online Teaching. Administered and led by the California State University Center, this international cooperative organizes a repository of high quality learning objects. Merlot provides sophisticated metadata and employs a peer review process in combination with user comments to assure quality.
ARIADNE KPS (1996)	The Knowledge Pool System represents the core of ARIADNE's infrastructure, coordinated by Erik Duval. This digital library organizes learning objects with sophisticated, multilingual metadata, i.e. LOM (2.3.4) extended with semantic metadata for science type, discipline, and sub-discipline. A concept navigator allows for browsing more intuitively.

**Table 10: Educational repositories collect interactive learning objects and work on challenges such as component granularity, reusability, and quality assurance.**

More than 50 applets [vanDam02] explore concepts in the areas of color theory, signal processing, scene graphs, lighting and shading, viewing techniques, texture mapping, and linear algebra (see Figure 5). Available software components mainly focus on Java 3D primitives (arrow, axes, cone, cube, cylinder, grid, sphere, etc.) and interactive 3D widgets (translate/rotate draggers). Users may run the applets with their Web browser (using the built-in Java Virtual Machine, or a Java plug-in) or browser-independently with Java WebStart. Applets and components are bundled as Java archive (JAR) files and can be downloaded for free.

The long-term *E-Slate* project [Birbilis00, Kynigos01] in Greece creates an easy-to-use visual component toolkit for teachers. Due to problems with software compatibility [Spalter03], they also develop a custom desktop, that is, a container environment (see Boxer, 2.2.2) offering integrated authoring and education. On the icon-driven desktop, users may interconnect components through visual “plug and socket” programming, which provides both a data flow and protocol (component dependencies) connectivity. An essential Logo **scripting** component extends built-in plug-in control facilities. Each component carries a set of scriptable primitives involving data passing, state changes, and event handling, all of them available through the desktop. Like in other builder tools, the user may directly manipulate component layout and appearance. Components are JavaBeans that must implement an additional E-Slate API. About 30 components span information handling and visualization tasks (database, map viewer, chart, vector), media handling (image editor, canvas, TV, Web



**Figure 5:** The Exploratories project at Brown University develops Java applets for teaching introductory Computer Graphics. Exploratories combine exploratorium and laboratory aspects, and provide graduated, multi-level component design. [Laleuf01]





**Figure 6:** The E-Slate project in Greece creates an easy-to-use visual component toolkit for teachers. A custom desktop allows for modifying component layout and appearance, and lets users control component primitives by scripting. [Birbilis00]

browser), simulation support (agent, turtle, stage, time, clock, variation), and common GUI controls (button, checkbox, menu, list, text, etc.). We still can infer the project's early plan, which was to create microworlds for learning Geography (see Figure 6). The chosen granularity balances usability and flexibility: users can modify components through visual programming and scripting only to some degree, further modifications must be resolved on source code level. Desktop environment, learning objects, and components are available free of charge.

Between 1998 and 2001, the *ESCOT* project [Roschelle99, Parnafes01] created component-based software for middle school math. Their declared goal was to investigate how software components may support the educational community in developing interactive learning objects; research focused on component reusability and interoperability. *ESCOT* reached a critical mass of components by collaborating with several companies (and saved time and costs [Spalter03]). They adapted and incorporated existing mathematics educational resources, such as Geometers Sketchpad, SimCalc, AgentSheets, EOE components (see below), and others; component granularity ranges from simple GUI widgets such as scrollers, sliders, number and text entry boxes to more complex, customizable components that such as a grapher, a spreadsheet, an histogram, a geometry sketching component, and agent-based simulation components. The Mozilla Rhino engine formed the base for a JavaScript scripting



**Figure 7: ESCOT demonstrated how component reuse supports the educational community in developing interactive learning objects. The project created middle school math problems using third-party components, and promoted them in the Drexel MathForum. [Roschelle99, MathForum03]**

component. Such component reuse minimizes implementation efforts – Sketchpad, for instance, brings with it all geometry functionality, maintains all geometry relationships, and provides drag & drop interactivity and animation. As it publishes parameters, measurements (e.g., the area of a polygon), and actions to the component’s API, ESCOT can modify and trigger them via standard Java components (e.g. a button). The main drawback of **third-party components** lies in intellectual property issues and licensing [Spalter03]; now that the project is finished, its public repository contains about 40 applets.

ESCOT promoted its work for use in real classrooms by designing the Drexel MathForum’s electronic Problem of the Week (ePoW, [MathForum03]), a well-established institution and process in which interesting, non-standard math

problems are posted to the international Web audience. Students work on the problems and submit solutions, and the MathForum offers feedback, help, and exemplary solutions. Finally, ESCOT evaluated the use of visual programming builder tools (2.2.2). They customized Sun’s BeanBuilder, a tool for connecting JavaBeans components, to support common wiring patterns and enhance connection flexibility.

There are also many repositories collecting **stand-alone** educational material. One of the first efforts in creating a community developing and using Java applets is the *EOE* [Azevedo01]. It organizes applets by areas and covers a large variety of subjects such as Computer Science, Social Sciences, Arts & Music, etc. (actually, most of them belong to the SMET domain). Applet metadata briefly describes object functionality, and provides source code for user modifications, member reviews, pedagogy (prerequisites, learning level, educational objectives, use time, form, structure, interactivity level), and technical comments. EOE’s primary trouble is that the repository collects only links to objects. In 2000, roughly 30% of their applets had broken links or other technical defects. Furthermore, they use a proprietary set of metadata providing only subjective tendencies instead of comparable concepts. For example, applet metadata may state “discovery” pedagogy, “hyperdimensional” structure, “high” interactivity, “image” presentation, and “math” relation. As EOE relies on free-form metadata, searches become practicable only for the licensing, source availability, and target education level. Such lacks reflect in community activities. In the field of Geometry, 8 out of 142 learning

object members have given reviews, with only 2 technical comments, and neither metadata nor pedagogy information. (Note that educational value and use of a given source code is not considered in metadata.) Searching or browsing becomes difficult. Not a single applet provides end-user accessible parameters through which we can tailor the applet even mildly. EOE applets are not based on software components; they exist as stand-alone, non-interoperable entities.

*MERLOT* [SmithGratto02] improves this basic strategy. Like EOE, the repository collects links to web sites assisting education in a variety of content areas. It arranges subject categories into subcategories, which makes searching/browsing for learning objects a lot easier. Material type is not restricted to Java applets, but ranges from simulation, animation, tutorial, drill & practice, quiz/test, lecture/presentation, collection, case study, to reference material. Regarding interactive learning objects, the repository features about 2250 simulations and 300 animations. Metadata for the primary audience and technical format (15 common formats, e.g. Java, Shockwave, Flash) is similarly detailed. Developers can attach source code to allow user modifications. MERLOT assures quality by employing a (five-star) peer **review** system and user ratings. Two higher education faculty members take part in the reviewing process, and a learning object must average three stars to become part of the repository. The review process covers three areas: quality of the content (accuracy, clarity of presentation, relevance), ease of use (interface design for faculty and students, engagement, interactivity), and effectiveness as a teaching tool (objectives, potential for integration into classes, instructional flexibility). About 950 out of 8800 learning objects come with peer reviews, among them 250 simulations. The MERLOT community is active: approx. 1850 users have submitted ratings and comments such as reports of activities they have developed to use in conjunction with the sites.

The sophisticated *ARIADNE* Knowledge Pool System (KPS, [Duval01, Duval02]) represents a digital library for learning objects with standardized **metadata** (2.3.4). Having a European background, multilinguality is one of KPS's key issues. ARIADNE metadata rearranges the LOM document hierarchy and introduces semantic metadata for science type and sub-discipline. Users may browse or search for science type by successively restricting available disciplines, sub-disciplines, main concepts, main concept synonyms, and secondary concepts. A concept navigator allows for browsing concept space. ARIADNE further develops tools for storing, searching, and retrieving their learning objects, e.g. the TM5 tool for test queries. KPS contains 1300 active learning objects (400 validated ones), whereas active objects denote courses, exercises, experiments, problem statements, questionnaires, self-assessments, and simulations. If we restrict searching to technical media (MIME) types, we find 46 Flash/Shockwave objects and 21 Java applets, mostly animations and low interactive objects in the SMET domain. Because metadata supports neither software components nor adaptability (e.g. in terms of Model View Controller, or by describing parameters and methods, or by including source code) produces stand-alone black boxes that can be used only "as is". Compared to MERLOT, we miss reviews and user feedback.

### 2.4.2 Software Components

So far, we considered content reuse as entities. Let us now address parts of interactive learning objects. Along the way, we have already met the building block design principle (2.1.1), messaging and visual programming (2.2.2), and projects developing interactive learning objects (2.4.1). Today, developers generally agree in basing development onto a toolkit of reusable software components [Klein98a, Laleuf01]. Roschelle [Roschelle98] states:

*“In our role as developers, we cannot afford build up high quality versions of each component we need from scratch. In fact, some components, such as computer algebra, are so expensive to build that we cannot afford to build them at all. [...]. [Component software architecture] could allow our development efforts to focus on narrow niches where we can make a unique contribution while allow our research efforts to draw upon a much wider collection of standard educational components.”*

Use of components redefines both programmer and educator tasks, as it “redefines the line between software creation and content authoring” [Yaron01]. Educational material is no longer seen “as is”, but as a learning object that can (and should) be adapted. In the same line, learning theories demand that the learner should be engaged in a construction process, i.e. creating and modifying software models to refine the learner’s own mental model (2.3.1). How many interactive learning objects did ever fit your own mental model of the represented topic? The great challenge is to create a matching set of graduated, interoperable software components for both programmers’ and educators’ use. Yaron explains why interactivity and reusability are, in some ways, opposing goals:

*“From a collections perspective, the level of interactivity required for engaging activities leads to monolithic chunks of software that are difficult to subdivide into components that promote adaptation and reuse.” [Yaron01]*

We define a **software component** as an internal element of an learning object that can be reused in other learning objects. The argumentation of component reusability follows the one for learning objects. However, only a handful of projects within the educational community deal with component issues [Spalter03]. Moreover, software components employed in an educational environment have different properties than all-purpose components – an educational software component must support teaching and learning per se. For example, understanding a concept might require studying a component’s interior (model). Content visualization (view) and interactivity (control) should match didactical goals. In addition, programming will generally favor understandable software structures (source code) over efficiency. Cunningham and Bailey, for example, illustrate the use of the scene graph structure for teaching [Cunningham01].

Any component software architecture (CSA) must deal with issues related to reusability (are components exchangeable?), extensibility (can we plug-in custom components?), granularity (smart, flexible components vs. ready-to-use application-like components), standards (will they interoperate with third-party components?), relationships (can the system update interdependent component parameters automatically?), and interactivity (are there built-in control structures that support direct manipulation?). Popular component standards are e.g. Microsoft COM (component object

model) or .NET (components with metadata), Sun JavaBeans (standard Java component model), Enterprise JavaBeans (server-side components, J2EE – Java 2 enterprise edition, ONE – open net environment), and Borland Delphi components (VCL – visual component library, CLX – component library for cross-platform development). They support component models such as data flow, data sharing, event-based messaging, and Model View Controller.

**Model View Controller** (MVC, [Krasner88]) deals with component interdependencies by decoupling a component's core data and functionality (Model), presentation (View), and user interaction (Controller). The Model notifies its Views when it changes and enables the View to query the Model's state. The View renders the contents of a model, accessing Model data and specifying its representation. It is the View's responsibility to react to model changes and maintain consistency in its presentation. The View forwards user gestures (e.g. button clicks, menu selections) to the Controller, which defines behavior; it interprets user gestures and maps them into actions to be performed by the Model. Note that MVC – separating an object's functionality, visual representation, and interactivity – is a Smalltalk design principle (2.2.2), and still the design pattern of choice for interactive Java applications [Singh02]. We will apply Schulmeister's interactivity levels (2.1.3) to the MVC pattern later (3.1), and extend MVC further to a more granular one regarding construction and interaction issues (3.2.1).

*Example (Model View Controller) Consider particles thrown into a vector field, e.g. flowing water. MVC would encapsulate geometry and physics in the Model components (particle position, velocity, acceleration, etc.), employ View components for graphs, tables, etc., and Controller components for user interactions, e.g. mouse or keyboard actions to change a mode, or parameter.*

Separating Model from View is essential for synchronizing multiple representations of shared data (2.1.2). In Java, dependencies are resolved by event-based **messaging**. Each View that is interested in changes of a Model creates and registers a listener object. As the Model is updated, it notifies each listener by sending a corresponding event object. JavaBeans components use such messaging to implement data flow; specialized events notify listeners of property changes, which might be vetoable (that is, listeners may prevent actions from occurring). Builder tools like Sun's BeanBuilder (see visual programming, 2.2.2, and ESCOT, 2.4.1) make it possible to design data flow visually by operating on JavaBeans metadata, which specifies publicly available properties, functionality, and connectivity.

Interdependencies comprising a bi-directional nature can hardly be managed by simple messaging. Remember for instance the geometry constraints in Sketchpad (2.2.1) or ThingLab (2.2.2). We require an additional mechanism providing tools for reasoning about messages and responses, in particular about the interactions among them; *Alan Borning* therefore introduces **constraint** components [Borning81]. A constraint represents a relation among components that must always hold. It is

specified by a rule and a set of methods for satisfying the constraint. More generally, a constraint can be seen as an algorithm component – Brown University’s algorithm animation system *BALSA* [Brown84] for instance uses such components for separating algorithm from animation issues, and the dynamic geometry software *Cinderella* [Kortenkamp99] takes a similar approach.

A more complex graphical presentation is represented by **scene graph** components [Strauss92, Bell95]. This directed, acyclic graph consists of a set of container components (group nodes), scene nodes holding geometry information, attributes (color, texture, etc.), camera parameters, and behaviors (interactivity and animation). Actions traverse the scene graph in order to render the scene, perform picking, calculate bounding boxes, etc. Until today, there is no built-in interaction mode for 2D/3D graphical scenes, neither in Java 2D, nor in Java 3D. Typically, the system (e.g. OpenGL) only supports picking, and developers build custom behavior toolkits to include interaction modes such as zoom, pan, rotate, walk, select, and drag & drop. We will discuss details of the scene graph structure later, when we compare classic approaches with our own extensions (3.2.3).

### 2.4.3 Scripting

*“[T]he principal challenge is moving the conception of component software from a developer-centric viewpoint toward a domain-centric viewpoint” [Roschelle99].*

Developers reduce the complexity of developing interactive learning objects by decomposing them into software components (2.4.2), not only to save time and cost, but also to enable end users to customize content, layout, and user interface. However, interactive learning objects and corresponding components are executable pieces of software, and difficult to adapt to the diverse needs of the educational community.

Many of the EOE’s applets (2.4.1) enable adaptation by including Java source code. Skilled programmers may then rework a learning object completely, using efficient data structures and low-level communication. However, Yaron [Yaron02] mentions that “even if source code is available, and the changes required to make the content useful in the particular classroom are small, instructors do not typically have the time or expertise needed to implement them”. Many other members of the educational community [Roschelle99] criticize the current excess of computer **programming** in developing interactive learning objects. Normally, educators with pedagogical expertise lack in technical background, which excludes them from the creation process of an interactive learning object. Papert’s approach therefore was to simplify programming (see Logo, 2.3.1). Kay’s working group set off a further shift towards direct manipulation, drag & drop construction, and visual programming (see Smalltalk, 2.2.2, and ESCOT’s use of builder tools, 2.4.1). Current tools such as ALICE [Conway97] reduce programming to graphical if-then-rules and automate common tasks. However, as Adele Goldberg, a member of Kay’s group and his later successor, mentions (as cited in [Roschelle98]), the principle problem remains: few educators have the time or inclination to become programmers at all.

We therefore turn to a complement model that seems to be more natural for authors that are not technically inclined: creating Web content and extending its abilities. Yaron [Yaron01] characterizes that approach as follows: Instead of viewing “the creation of curriculum material as primarily a programming task, and simplifying programming to the point where it becomes accessible to instructors [...], our approach starts from a very different perspective, that of curriculum creation as Web authoring”. He argues that most members of the educational community are already familiar with **Web-based authoring** such as using hypertext links or image maps. Basing authoring on the hyperlink paradigm has – from an author’s perspective – advantages compared to simplified programming. It is straightforward to go from creating a hypertext link to sending a message to a learning object. Instead of linking to an entire object, we might introduce links to, from, and between software components (or other internal items or substructures).

Such an interlinking with an object’s substructures requires a mechanism specifying link anchors and targets independently from the object’s content type, content, and structure. Remember the *Dexter Hypertext Reference Model* that regards the object’s interior (‘within-component layer’) as being outside of the hypertext model per se (2.2.2). Instead, it proposes indirect addressing by a pair of anchor identifier and value. While the identifier provides a constant, unique referent for linking, the anchor value specifies some item or substructure within an object. Obviously, only the application responsible for handling the object can interpret the anchor value; as the object changes over time, the anchor value will be changed to reflect structural changes or movements of the item to which the anchor is conceptually attached.

Technically, browser plug-ins render the Web interactive (2.2.3). The communication model between plug-in content and hypermedia environment seriously restrains interoperability. Consider today’s interactive Web content: does it show adequate hypermedia embedding or rather black box behavior? In the case of Java applets – which ironically start up as gray boxes – authors may access all public API functionality of the object by the use of **scripting**. Scripting languages like AppleScript, JavaScript, or VBScript (VisualBasic) allow for lightweight programming [Roschelle96]. They offer an object model of the current application and facilitate control of object behavior within and across hypermedia objects. Authors embed Web scripts as plain text into HTML content. A Web browser initiates and starts scripts either when opening a Web page or after specific user interactions such as button or mouse-over events. The other way round, scripts may also process messages (events) from software objects, e.g. to synchronize objects, or give context help.

A first approach therefore is to define public methods and properties of an interactive learning object that then may serve as anchor identifiers. *Wolfgang Christian* implements this idea in his *Physlets* (applets for teaching physics) toolkit [Christian00]. Each of his components ships with a predefined set of properties and methods that we may use in scripting. Christian further splits learning objects into many small, but interconnected components (applets) that can be distributed all-over a Web



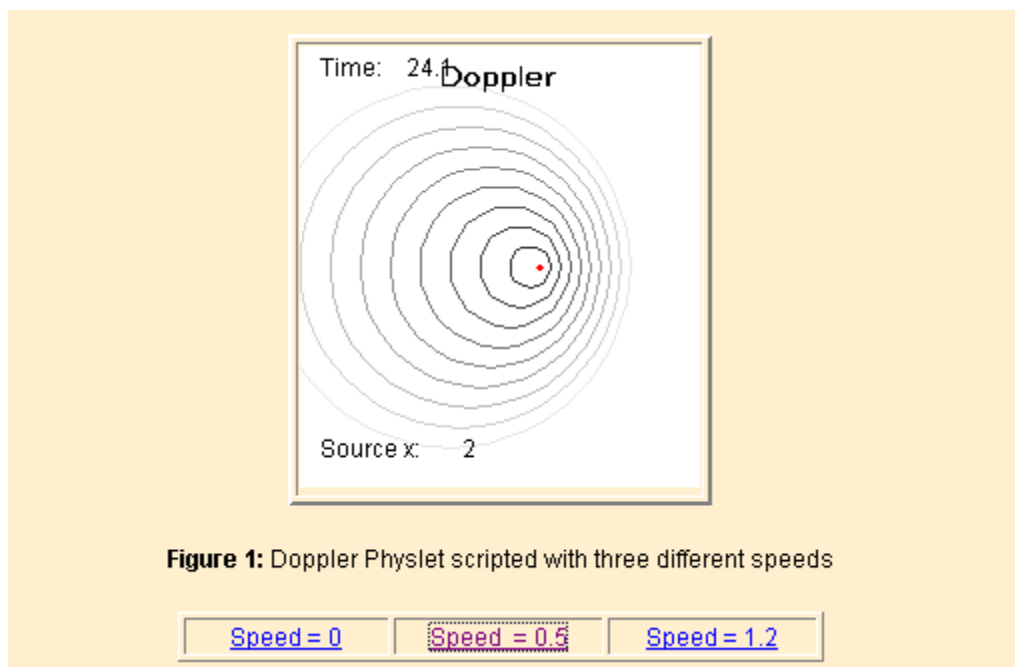


Figure 1: Doppler Physlet scripted with three different speeds

**Figure 8: Christian’s Physlets represent small, interoperable components for teaching physics. They offer predefined, scriptable properties and methods. Working as applets, physlets can be distributed all over a Web page, and still share data and communicate. [Christian00]**

page. This allows authors to interlink, for example, non-interactive, instructional course text with interactive, constructive microworlds. (Horwitz [Horwitz95] calls a microworld (2.3.1) that is connected with its hypermedia context by scripting a **hypermodel**.) However, we cannot restrict a Physlet’s given set of public functionality. Roschelle mentions:

*“The more functionality and flexibility an application offers, the greater is its usefulness. On the other hand, we only want to use a constrained set of carefully tuned features that can help students focusing their work.” [Roschelle96]*

The Physlet component granularity is not graduated. While each Physlet component represents an application, internal items or substructures are not encapsulated into components. Functionality that was not foreseen must be implemented by a new component. *Jeremy Roschelle*, who concentrates on flexibility in authoring mathematical representations and manipulating the object’s user interface, made a first step towards more graduated, scriptable components. His *MathWorlds* [Roschelle96] project enables users to control interface issues with AppleScript scripts, record scripts, and attach scripts to the interface. *ESCOT* and *E-Slate* (2.4.1) consequently encapsulate scripting functionality, i.e., a script parser and interpreter, into corresponding scripting components (JavaScript, respectively Logo) – that way, developers can adjust publicly available functionality as needed. Lastly, Roschelle mentions the use of so-called factored programs, which encapsulate MVC’s direct manipulation controllers, to generate scripts on user interactions.



## 2.5 Conclusion

This first part of this thesis concerned the nature of interactive Web-based courseware. We described interactivity in terms of direct manipulation, physical GUI ingredients, cognitive process, and quality. A combined definition will be given in the second part. We contrasted the WIMP-designed Desktop with the Web, which is limited to the hyperlink and plug-in technology, and portrayed efforts to render the Web valuable for education by focusing on structure, interactive multimedia, collaboration, and personalization.

While a start has been made with learning management systems and metadata standards, great interactivity is not considered in current learning technology. We found that current interactive learning objects act as black boxes in a Web-based courseware, which can be neither adapted, nor interlinked in a fine-grained manner. Development swallows a lot of resources, and resulting software cannot be reused. We therefore outlined approaches to software components and scripting that promise to provide adaptability and interoperability on all software levels. However, many questions are left open. Can we conceptualize and realize a multi-level, scriptable, component-based architecture? If so, how do we manage learning objects, software components and scripts in a Web-based courseware, and allow community members to work with them? We will give appropriate answers in the second part.



### **3 GRIS/ILO Interactive Learning Objects**

3.1 MVC Interactivity.....	60
3.2 Software Components.....	61
3.2.1 The ORC-SG Design Pattern.....	61
3.2.2 Object, Renderer, Constraint (ORC).....	62
3.2.3 Scene Graph and GUI (SG).....	65
3.2.4 The Toolkit.....	70
3.3 Adaptability and Interoperability.....	71
3.3.1 Scripting.....	71
3.3.2 Networking .....	75
3.3.3 Scripting Database .....	77
3.3.4 Drag & Drop Scripting .....	79

### 3.1 MVC Interactivity

We already clarified the denotation and relevance of interactivity in learning by emphasizing successively the physical, perceptual-cognitive, and communication-theoretic points of view (2.1). Let us now combine the major concepts of interactivity and reformulate the terminology in terms of Computer Graphics principles, particularly in terms of the Model View Controller (2.4.2). We propose a so-called MVC Interactivity [Hanisch03b] that provides a precise definition of highly interactive learning objects. In our notion, a highly interactive learning object allows for directly manipulating its view, parameters, and functionality.

At first sight, MVC meets technological, didactical, and cognitive demands. The paradigm separates interface issues (Controller, e.g. direct manipulation) from an object's visual appearance (View) and its state or functionality (Model). However, parameter interactivity differs in its symbolical meaning from interactivity on the structure/model layer. We therefore break down MVC's Model into parameters and internal functionality; while the first represents an object's state, the latter comprises the underlying structure and components. Note that Schulmeister's understanding of an object's model refers to its functionality, that is, to its structure and components (object parts), and the object's state corresponds with his notion of parameters (2.1.3).

We consider an object a **highly interactive learning object** if it provides means for

1. representation of domain knowledge that may induce a learning process
2. illustrative actions
3. direct manipulation of object view
4. direct manipulation of all essential object parameters
5. direct manipulation of object functionality (structure, components)
6. adequate feedback and help

We integrate all levels of Schulmeister's taxonomy, require direct manipulation in all aspects, and explicitly ask for a proper design of interactivity with respect to the range of interactive parameters. Note that we strive for great interactivity only – other degrees of interactivity can be derived. Point 1 formally defines a learning object (2.3.4).

We imply that a highly interactive learning object visualizes complicated topics and relationships graphically and allows for direct manipulation of all parameters belonging to the topic's core. Learners can modify internal components and get visual feedback or help wherever needed. Modifying the model may require visual scripting (3.3.4) or visual programming (2.2.2).

## 3.2 Software Components

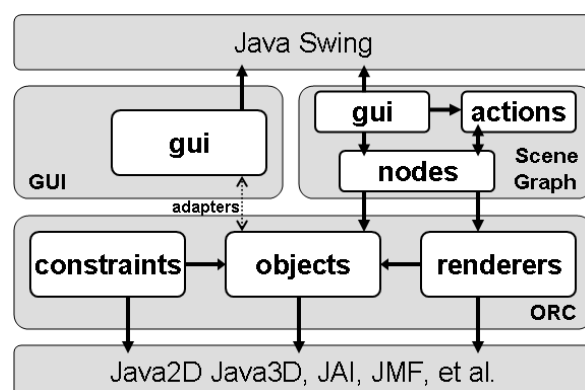
### 3.2.1 The ORC-SG Design Pattern

Developing hundreds of highly interactive learning objects naturally leads to a component-based architecture supporting MVC, constraints, and a scene graph structure (2.4.2). MVC separates an object's functionality, appearance, and interactivity, constraints encapsulate component interdependencies, and a scene graph represents complex graphical scenes. We portrayed how to create and modify learning objects by connecting components (programmatically or visually).

However, there is no design pattern combining such component types. While MVC is included into popular component standards, constraints still require custom structures. The scene graph exists as a separate, incomplete architecture (for example, we have Java 3D, but not a 2D scene graph), requiring custom extensions, e.g. to provide adequate interactivity. Moreover, we have argued for a multi-level architecture enabling us to adapt, combine, and exchange components on all levels. We therefore propose and implement a more granular MVC component model with respect to construction and interactivity, which we call ORC-SG (object, renderer, constraint, scene graph, GUI) [Hanisch03a].

Our implementation is based on Java 2 (see Figure 9). To maximize usability (and minimize netload), we heavily used the included packages Java2D and Java Swing, and, as appropriate, the standard packages Java Advanced Imaging (JAI), Java Media Framework (JMF), and Java 3D – packages providing basic functionality for mathematics, geometry, visualization, and user interfaces. Our own packages extend this framework, respectively with components encapsulating objects, constraints, renderers, GUI, and a scene graph structure. The latter ones include sub-packages for a scene graph's nodes, actions, and GUI.

We do not ship interactive learning objects with all these components; instead, we bundle components into packages (Java archives, JAR) and specify required packages in the learning object metadata. Browser or plug-in may then reuse formerly downloaded (cached) packages, and the learning object's actual netload shrinks to a few kilobytes. To avoid complex component interdependencies between different packages, we set up layered packages containing only one-directional dependencies, and use **adapter** components with standard Java interfaces like JavaBeans properties.

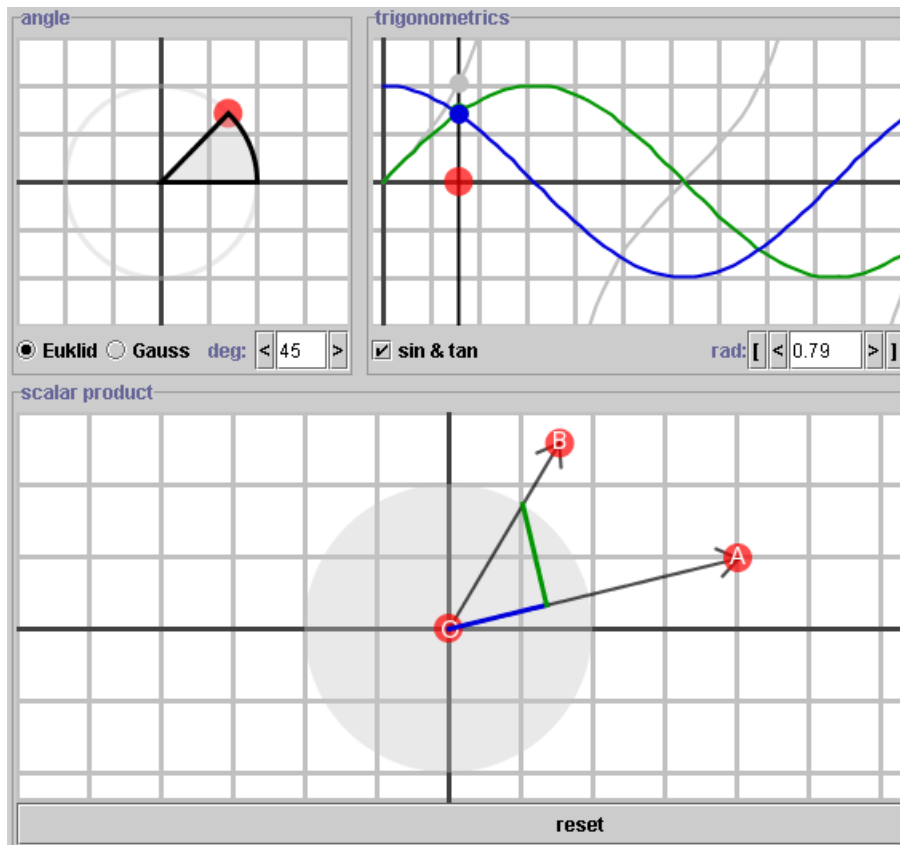


**Figure 9: The ORC-SG design pattern for a component-based software architecture. Front- and back-end of the highly interactive learning objects we developed are standard Java packages.**

### 3.2.2 Object, Renderer, Constraint (ORC)

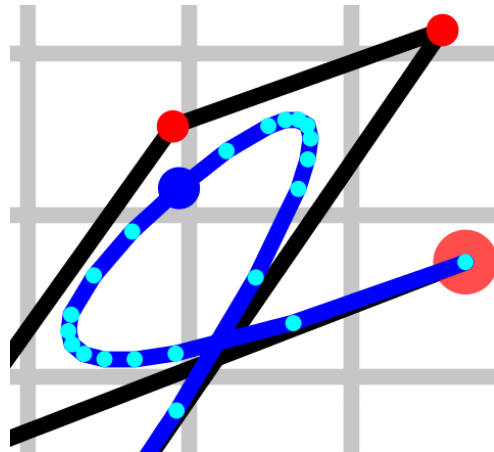
Usually, MVC's model encapsulates both state and functionality of a component. As educators, we are also interested in its construction. Consider Bézier curves – how would we teach them without the constructive algorithm of de Casteljau? Therefore, we further separated constructive information into a **constraint** component (e.g. orthogonal lines, a point on a curve, 2.4.2). A constraint automates necessary parameter updates, which is useful in the case of many interdependent parameters (see Figure 10). Before performing the update, we announce parameter changes to all listeners registered for the given type of modification, and let them accept or veto the change, or constrain data accordingly. The update phase is restricted to the ORC layer; further updates involving SG components must either be promoted programmatically (to avoid rendering visual information multiple times) or be handled directly within the specific component (e.g. in response to a user input).

Instead of implementing a constraint solver, we simply rely on a **data flow** model (forcing the author to avoid circle definitions, which might otherwise lead to deadlocks). Our result are reusable and exchangeable algorithms – which is useful for more than just teaching alternative computer graphics algorithms.



**Figure 10: Constraints encapsulate constructive information. In this example, they manage bi-directional, direct manipulation of all essential parameters, while synchronizing dependencies automatically.**

Our **Object** components store state information and primitive functionality. They fit smoothly into the Java 2D/3D component hierarchy, e.g. point components derive from the corresponding Java component, and abstract point functionality supports both dimensions. Our geometry objects (point, vector, mesh, etc.) contain a transformation cache to avoid redundant recalculation of transformed states.



*Example (Curve Objects)* All of our parameter curve objects, from simple point sets to Taylor/Lagrange/Bézier/-BézierSpline/BSpline curves to arbitrary functions (our parser component supports standard math and trigonometric functions), derive from a Curve component. This abstract base component provides functionality for querying the point list, performing equality and epsilon tests, calculating derivations, bounding box, length, distances, etc. A flatness parameter controls curve approximation by a discrete number of points; we perform recursive interval bisection until the resulting points' distance is less than the given flatness (see Figure 11). For each affine transformation operating on the curve, we store the transformed discrete points in the transformation cache (a simple hash table). Derived components implement case-dependent functionality such as single point queries or replaces functionality such as the general difference method for calculating derivations by optimized derivations, e.g. for Bézier curves or explicitly given ones.

**Figure 11: Geometry objects hold state information and primitive functionality, together with a transformation cache. All of our parameter curves derive from an abstract curve providing functionality like curve approximation for a given flatness. Our abstract curve renderer in turn draws curves output-sensitively, using polylines and one-pixel default flatness.**

In contrast to this, our **renderer** components hold an object's view – its visual information. This enables students to change views and focus on teaching content in programming, without having to deal with technical output issues (see Figure 12). In MVC, it is the view's responsibility to react to model changes and maintain consistency. In ORC-SG, renderers are passive components; the scene graph will perform all necessary synchronization. Renderers may hold arbitrary **properties** to describe their appearance (e.g. colors, line strength, strokes, or fill type). The renderer's base component stores them as string-value pairs in a hash table and allows for a scripting style like `renderer.setProperty(inactive-Color, Color.black)` independent of the actual renderer type. A property cache ensures efficiency: we cache properties additionally in the derived renderer, and retrieve them from the hash table only if they become

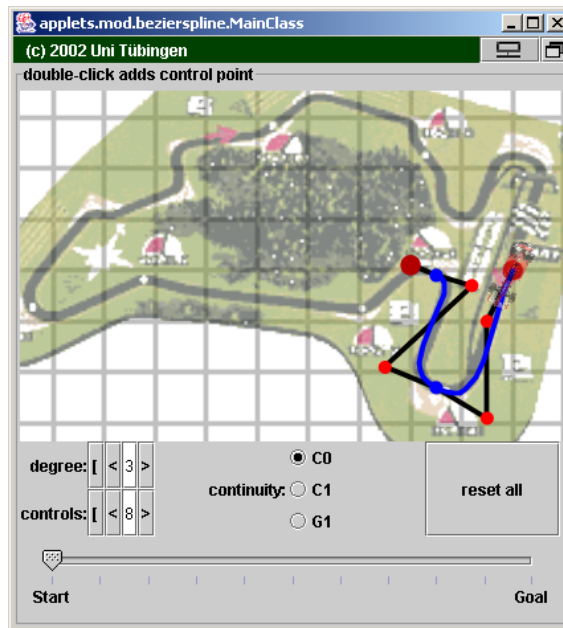
invalid. As users wish to select an object's visual shape instead of geometry, renderers further hold selection functionality such as picking with a screen point or an arbitrary shape (e.g. a rectangular selection).

*Example (Picking)* Our string renderer provides both a 'pickable' and a 'pickInside' boolean property. If merely the first property is set, we let users pick the string's bounding box on screen. While this is useful for small-sized text, such picking behavior might confuse users when dealing with larger-scaled text. There, they would rather expect to pick only visible portions of the string, that is, the inner area of the characters, – which is, in fact, the behavior that our second property enforces.

*Picking with an arbitrary shape is similarly resolved per character. We test each character's midpoint, and pick the string only if half the characters lie inside.*

All of our renderers render into the standard Java Graphics2D object. That way, we can output into any Java GUI component, or render into any other component providing a Graphics2D object, like the Java Image component. Many of our renderers further work **output-sensitively**, a necessity for fine-detailed graphical scenes such as grids, point clouds, or vector fields. Just imagine setting a small zoom factor for vector fields while keeping grid size constant – rendering would slow down immediately.

*Example (Output-Sensitive Rendering)* Our curve renderer contains properties such as line strength, strokes (e.g. dotted, textured), point color, inactive and active color (most of our renderers visualize objects in two states, inactive/unselected and active/selected), a pickable flag determining if users may select the object, and a flatness parameter (with one-pixel default, see Figure 11). We render the curve into a Graphics2D object (which, in the simplest case, belongs to a graphics canvas) bringing with it an affine (canvas) transformation, clipping bounds, and several rendering hints, e.g. antialiasing, dithering, or interpolation. Curve flatness results from calculating the required flatness \*before\* the canvas transformation meeting the given rendering flatness was applied. We query the curve points, perform line clipping,



**Figure 12:** Encapsulating visual information into renderers enables students to change views and focus teaching content in programming. Here, students pilot a racing car to the goal by programming and connecting a spline curve's Bézier segments with C0, C1, or G1 continuity, without having to deal with technical output issues.



*and finally draw a polyline using the given stroke. Note that affine canvas transformations (zooming, rotating, translating, etc.) will produce different point sets due to varying curve flatness; however, we have to perform interval bisection only once, and then pick points in the bisection tree accordingly. Only scene transformations operating on the curve will require us to re-approximate the curve.*

Objects, constraints, and renderers form a self-contained package. As the number of possible constraints and renderers is unlimited, a **plug-in architecture** allows for incorporating additional, custom ones. Our plug-in mechanism retrieves components by Java Reflection and naming conventions: we search for components with predefined syllables in a list of default locations, try to retrieve the class file dynamically via Java Reflection, and create an instance fitting to the given set of parameters. Users may extend the lists of locations and syllables. Third-party components can be included in supplementary packages.

*Example (Plug-In Architecture) We create a geometry constraint for orthogonal lines with the statement `Geo. constrain("orthogonal", outLine, inLine, inPoint)`. The plug-in mechanism scans all given locations and syllables, retrieves a fitting `OrthogonalConstraint2D` component in the package `grdev.geo.objects2D.constraints`, and instantiates it. The constraint immediately connects to the input parameters `inLine` and `inPoint`, and updates the output parameter `outLine` to become orthogonal to `inLine` through `inPoint`. We locate renderers similarly; each geometry object possesses a default renderer specified by naming conventions, e.g. we render a `Point2D` component by default with a `Point2DRenderer`. Note that the string-based approach integrates seamlessly into a scripting architecture (3.3.1).*

### 3.2.3 Scene Graph and GUI (SG)

Two other packages consider user interactivity. While a scene graph package provides advanced interactive visualizations, a GUI package extends the standard Java Swing component set with our own GUI components.

We developed specialized **GUI** components for reoccurring setups in animation (play, pause, forward, backward, etc.), file operations (e.g. file browser, image browser), layout (pre-defined, bordered panel groups), or informational tasks (e.g. progress level). They supply a consistent **look & feel** (2.2.2) and circumvent common pitfalls like concurrency in multi-tasking (threads), local activities vs. networking or applications vs. applets (class path, data access, security), and packaging (data access in JAR files).

*Example (Scalar GUI) We implemented a text field interpreting its input as a scalar value. A simple command in the form of `'scalar.setComponent(textField)'` connects this text field to a corresponding scalar component; from now on, these two are kept synchronized. Developers may optionally define the number of decimal places, lower and upper bounds, and increments, which inserts buttons accordingly. The increments define an adaptive speed-up behavior (initial delay, fast forward/backward) enabling users to keep buttons pressed (see, e.g.*

Figure 12, bottom left). Besides, remember a common pitfall when using floating-point arithmetic: it is rarely exact. The increment 0.1 for example would need an infinitely recurring binary (and would produce results like 1.000000000000001 instead of 1). We therefore employ Java *BigDecimal* functionality providing anticipated rounding behavior.

All of our GUI components are based on standard Java functionality. Remember further the **adapter** mechanism resolving component interdependencies between packages. In the scalar GUI example, an adapter translates the scalar component's parameter change events into less efficient, but standard, JavaBeans property change events, and vice versa. (Technically, the adapter components is registered as parameter change listener to the scalar component, and as property change listener to the GUI component. It reacts by firing a corresponding property change event, respectively a parameter change event).

The **scene graph** package provides default interaction behavior and visualization for our objects. This reduces technical issues and enables authors to concentrate on an learning object's didactical value, e.g. by enriching mathematical theory with concepts of digital storytelling (see Figure 12). A scene graph consists of nodes, actions, and a scene graph GUI. **Nodes** are made of at least one pair of geometry objects and matching renderer; we assemble them into a classical scene graph hierarchy (2.4.2) representing all information of our graphical scene. For performance reasons, we permit multiple occurrences of a node in the tree. Usually, our scene nodes contain an additional interaction sub-tree, which assembles several basic scene graph nodes to provide a desired interaction behavior. A first benefit is that we may reuse already implemented – and familiar – functionality. Nodes that must render text (e.g. labels in a coordinate system) can integrate labels with a string node; as our string nodes can be drag & dropped, users may use this feature to rearrange labels to fit their needs. A second benefit is the fact that we now can implement different interaction modes by providing different sub-trees. Think of interacting with a 2D circle: while one sub-tree might offer users to directly manipulate circle parameters by three interactive points on the circle's boundary, a second sub-tree might offer only two points, center and radius, for modification.

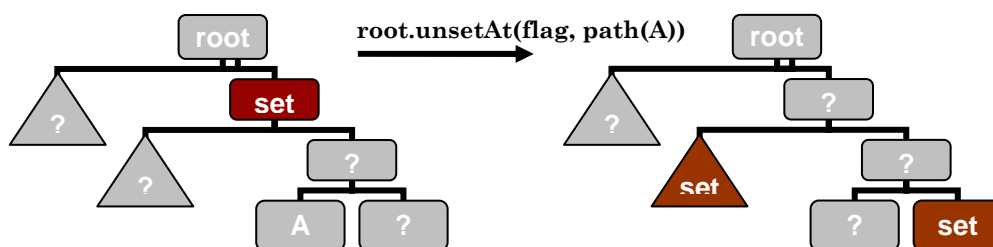


Figure 13: We accompany the scene graph with a global state tree (left side) holding arbitrary flags (visible, selected, restricted scene actions, etc.). Setting a flag at a given path indicates that it is also set at all nodes below. Unsetting a flag involves checking the nodes on the path and, if a node is set, propagating the flag to all children (right side).

Note that our 2D scene graph contains two different types of transformation nodes: an object and a canvas transformation. This corresponds to the camera node and object transformation known from classic 3D scene graphs. We included this differentiation to make use of object caching (3.2.2). The first class, object transformations, operates on geometry objects; their results are cached within these objects. The second class merely transforms the canvas we draw on, i.e. it represents alternating global views, or interactivity that globally zooms, rotates, or pans the scene. Canvas transformations can be stored directly in the standard Java Graphics2D object, which optimizes rendering.

Arbitrary scene graph **actions** traverse the node hierarchy in order to render the scene, pick some objects, drag them, and so forth. Simple models reflect the current state during scene graph traversal by a global push/pop state. Such an approach is not suitable for highly interactive graphical scenes. We replaced the state object with a global state tree holding arbitrary state flags. Flags mark nodes located at specific scene paths as, for example, visible, selected, or active for specific actions only. We optimized this state tree straightforwardly by saving shared states of siblings in their parent node (see Figure 13). While initially the tree is collapsed to a single node, we insert marked nodes successively whenever needed. State tree functionality consists of node primitives (set, unset, reset, isSet) and node operations specified by paths (setAt, unsetAt, isSetAt). While setting a flag at a given path indicates that it is also set at all nodes below, unsetting a flag involves checking the nodes on the path and, if a node is set, propagating the flag to all children. If all nodes become marked, the state tree's size becomes equal to the scene graph's tree size; therefore, interactive mass scenes would require further optimization.

Our 3D nodes use Java 3D functionality. Java 3D encapsulates functionality for non-static scene graph interactions into so-called **behaviors**. However, the built-in collection (billboard, LOD, interpolators)

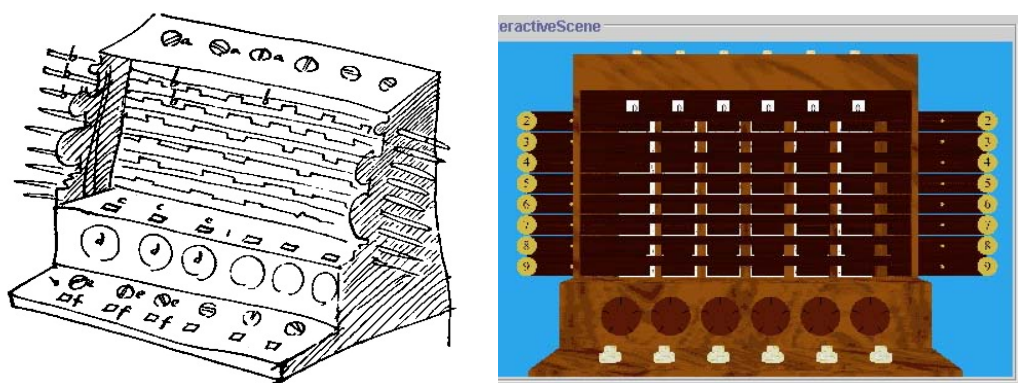


Figure 14: Our reconstruction of the Schickard calculator makes heavy use of the scene graph structure. We used a drawing found in Kepler's letters (left side, [Löringhoff78]) for modeling, and created Java 3D components with corresponding behaviors. Users may directly manipulate buttons, sliders, and gears, and obtain context help when moving the mouse over them.

does not cover the ability to react adequately to user input. We had to develop custom mouse and keyboard input behaviors to provide interactivity with both single scene nodes (scaling, rotating, and translating not only the entire scene, but also single nodes within the scene), and constrained behaviors (picking with subsequent animation, movements along some axis, periodical behaviors, etc.) [Hanisch01b].

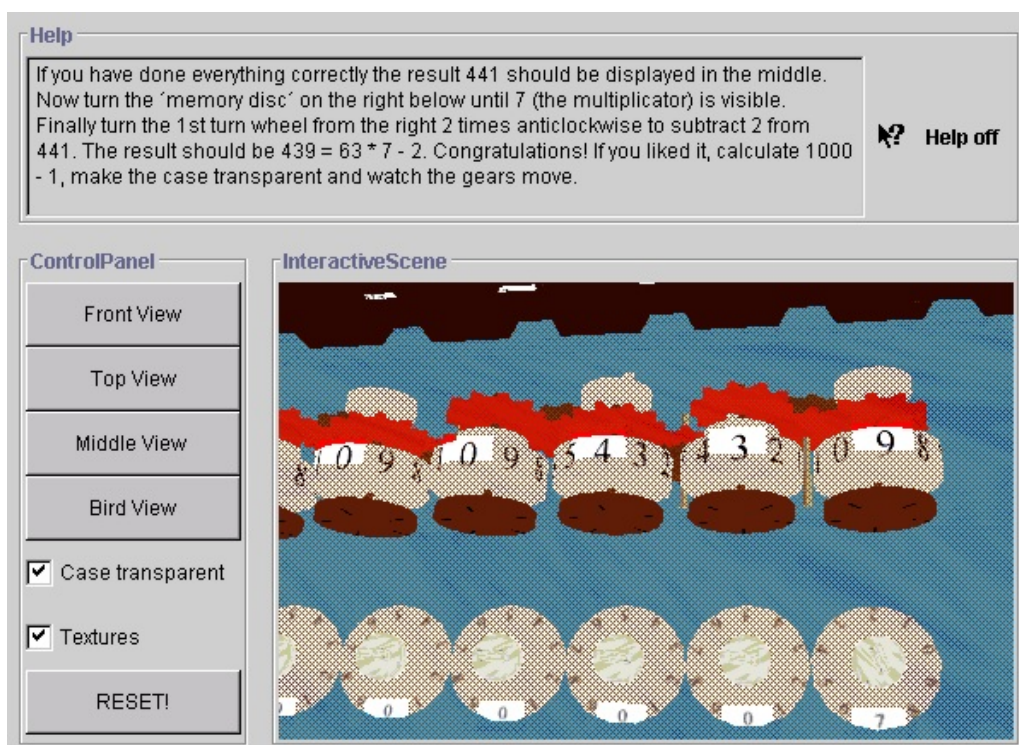
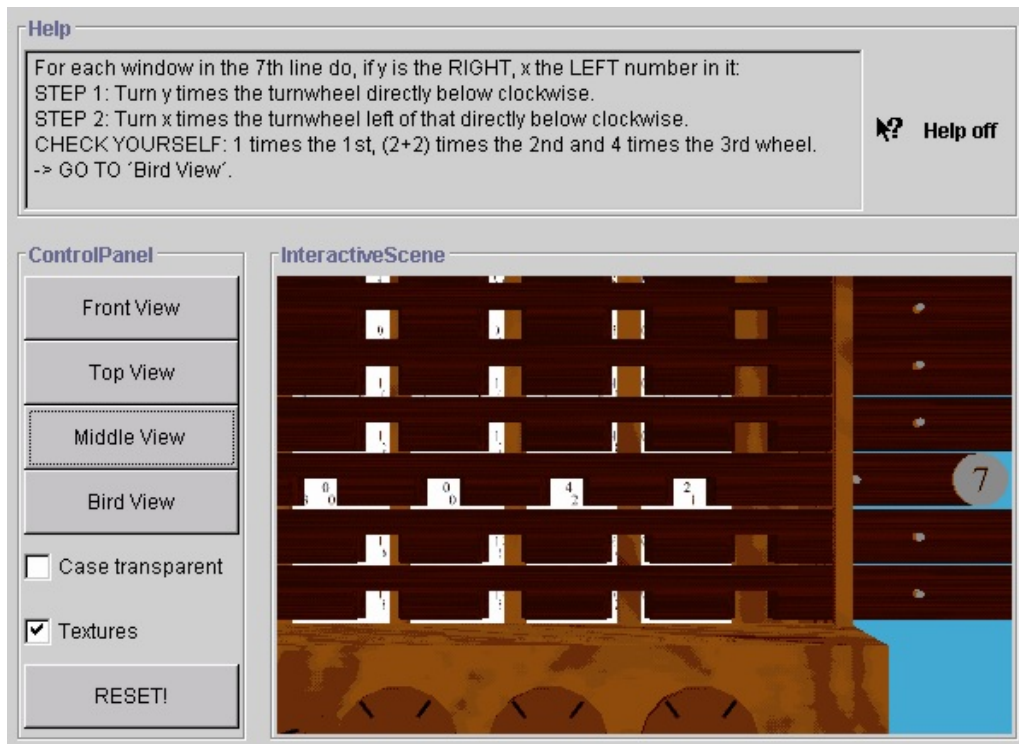
*Example (Schickard Calculator): Wilhelm Schickard, appointed professor of oriental languages, astronomy, mathematics, and geodesy at the University of Tübingen, invented the first known four-species mechanical calculator to add, subtract, multiply and divide. In 1623, he wrote to his dear friend Johannes Kepler, "[...] what you have managed to calculate by hand, I've tried to perform in mechanical ways in the last days and have constructed a machine consisting of eleven complete and six garbled gearwheels. It calculates instantly and 'automatically' (this word was written in Greek letters!) given numbers: adding, subtracting, multiplication and division. You would burst out laughing if you would be here and live to see how the left digits, if it goes over the tens or hundreds, increase on their own or while doing a subtraction taking something off" [Löringhoff78]. Some hundred years later, the faculty of Computer Science in Tübingen was named after Schickard.*

*Our reconstruction of the Schickard calculator [Hanisch01b] employs a scene graph and enables users to perform calculations like Schickard did, watch the calculator from any viewpoint and even gain an insight view of the 17 gearwheels (see Figure 14). Users may follow a trail explaining the handling, or obtain context help (see Figure 15).*

*Reusing slider, button, gearwheel, and cylinder components speeded up the scene modeling, and economized both CPU memory and disk space. While we implemented global interactions such as zooming, rotating, panning, and picking parts of the scene using standard Java 3D components (in our case the buttons, wheels and sliders), constrained interaction behavior required for the 8 slider movements and for the rotation of the 6 white gear cylinders (so-called Neper cylinders) took some more effort. We represented them as picking behaviors with subsequent animations (the cylinders), left/right movements (the sliders), or periodical behaviors (the turning of a gearwheel that stimulates its neighbor to turn every 10th step).*

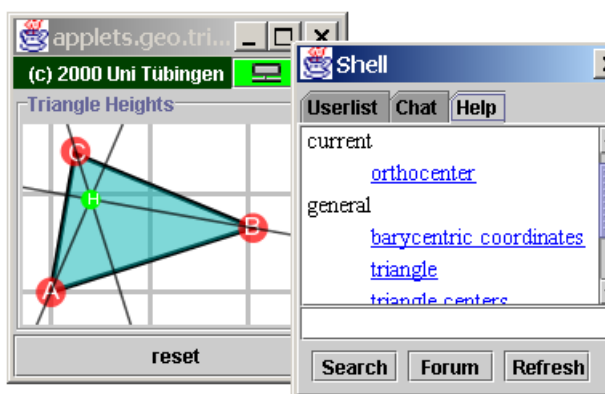
Finally, the scene graph package contains an appropriate **scene graph GUI** component, which maps user gestures and keyboard input into corresponding scene graph actions and resolves appropriate rendering actions. Compared with MVC, our GUI components and our scene graph GUI inherit all controller tasks.

We make further use of the scene graph structure by labeling scene nodes with **metadata**. That way, we realize context-sensitive references from a learning object's internal items to its hypermedia context. For example, consider a learning object offering direct manipulation of a triangle's special point (see Figure 16): according to its location, we expect context help respectively for the incenter, centroid, circumcenter, or orthocenter (there are more than 400 known triangle centers).



**Figure 15:** In exploring our reconstruction, the user may perform calculations like Schickard did, watch the calculator from any viewpoint, and even gain an inside view on the 17 gearwheels by making the case transparent and watch the gears move.





**Figure 16:** By labeling scene nodes with metadata, we are able to offer context help. The hyperlinks lead to corresponding passages of the course text.

Therefore, before offering help or references within a learning object, we request metadata of all currently selected scene graph nodes and GUI components with keyboard focus. We then collect, hierarchically bottom-up, all classifications within the scene graph and create hyperlinks to similar classified hypermedia elements. As we will illustrate later (3.3.3), authors have to describe all metadata appropriately.

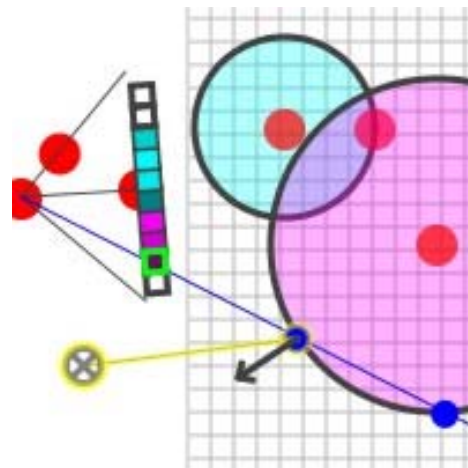
### 3.2.4 The Toolkit

Most of our interactive learning objects visualize algorithms and offer interaction with all essential parameters of an algorithm. For that reason, we identified reoccurring basic components for containers, data structures, 2D/3D geometry, images, video, and physical quantities [Hanisch00a].

We require a **container** model to create component composites and express hierarchy naturally (2.2.2). The Java language brings with it a Container component serving as a base component for windows or any other component occupying space on screen. Sadly, the embedding of browser plug-in content is incompatible with the Desktop's container model, and technology restrains us from bridging this gap (2.2.3). However, we are able to remove some obstacles by offering a base component for interactive learning objects, which enables users to toggle between document embedding mode (an embedded Java applet's position on the page is fixed) and overlapping windowing mode, and run the object as a stand-alone application or applet, either locally or online. Our custom resource loader decides on the correct sequence of local and network locations to be searched for, and bypasses annoying security issues that would arise in a naive implementation (for example, the default resource loader is limited to the applet's server). Finally, the base component displays startup and copyright information, and contains scripting (3.3.1) and networking (3.3.2) functionality.

By building composites of standard Java components and our ORC-SG components (using the Java Container or scene graph containers), we reach a **graduated** component hierarchy, i.e. a multi-level component architecture (2.4.1). While components of the ORC-SG layer represent universal, core elements of our architecture and our final, self-contained learning objects belong to a specific domain, composite components typically show up a granularity in-between; though they belong to a

specific domain, they are still universal, basic components. We have developed more than 130 interactive learning objects in the field of Computer Graphics, Geometric Modeling, Computational Geometry, Imaging and Video Processing, and Scientific Visualization (see case studies, 5) using such a toolkit of basic components. We developed toolkit extensions first, before any actual learning object; in fact, we spent most of our time on the former (which is typical for component software). We identified components for visualizing and interacting with data structures (list, graph, tree, etc.), geometry (camera, screen, grid, etc.), images and video (viewers, filters, etc.), and physical quantities (scalar, vector, field, light, etc.). Scene graph composites (so-called nodekits), plain text components, and image components provide direct manipulation and drag & drop functionality.



**Figure 17: Development of a toolkit of basic components typically precedes the development of an actual learning object. We created this visualization of the raycasting algorithm by connecting basic component composites (here, scene graph nodekits). All visible parameters can be manipulated directly.**

*Example (Basic Components) The learning object in Figure 17 demonstrates how we visualized the idea of raycasting. Several scene graph components implement the camera, the screen (a 1D pixel array), the objects (two circles), a point light, grid, and canvas. Using the standard dragging behavior, all visible parameters can be manipulated directly. The screen component listens to camera modifications and automatically adjusts its parameters. Similarly, the circles listen to the camera ray property and send the intersection information to the light source, which calculates the lighting model. Finally, the screen component sets the pixel hit by the ray to the color calculated by the light source. We reach a proper visualization by animating the camera ray property, that is, by looping through all values.*

### 3.3 Adaptability and Interoperability

#### 3.3.1 Scripting

Concerning reusability of interactive learning objects, we strive for (1) adaptability in design, layout, and functionality, and (2) interoperability with other objects (2.4.3). Our ORC-SG design pattern encapsulates matters of visualization, construction, and interaction into software components. However, interactive learning objects still appear isolated within a courseware, and modifications require low-level programming. Let us now focus on an appropriate scripting architecture enabling end

users to modify and exchange all components from within their Web environment, and interlink them with other ones.

With publishing languages (X)HTML and Java, a natural choice for a scripting language was JavaScript. We equipped our base component for interactive learning objects (3.2.4) with a single, publicly accessible scripting method; all scripting actions are delegated to a **scripting component**. Currently, our parser/interpreter prototypes cover only a subset of the JavaScript (ECMA) grammar. Scripts may import user-defined classes, instantiate objects, and call their methods. We further modified our GUI components to execute scripts on user actions. While a simple button component resolves standard script sequences, we simplified script syntax for multiple-choice components such as check boxes (a true/false value), combo boxes (one out of a list of string values), radio buttons (an integer value), etc. Scripts may therefore contain variables, which are replaced dynamically according to the user's choice.

The interpreter falls back on Java Reflection to create components or call their methods. As all scripts are located client-side, we obtain a performance overhead only in Java Reflection's search for constructors and methods, and, of course, in any deferred loading of required resources (classes, images, and data). A custom class loader component assures that our interpreter retrieves new components at first from (already cached) Java archives (JAR files), then from default plug-in paths, and then from given Web locations or local file systems.

End users may now create an entire learning object using an 'empty' base component plus an initial script. In practice, it is more likely that programmers code a learning object's initial functionality traditionally, and carefully declare a set of parameters as **script instances** afterwards. That way, developers can hide specific internal details, as well as filter publicly accessible information. The scripting component stores all script instances in a hash table; in contrast to the Java garbage collection mechanism, we explicitly permit users to remove instances. Scripts using an instance not yet created (which typically occurs with badly arranged script sequences) are ignored. Besides, note that it is not the browser who handles invalid scripts, but our scripting component. Browser-side scripting would immediately report invalid scripts to the user; some browsers would also disable further scripting. Our approach enables the learning object to decide which reactions should be accomplished autonomously.

We classify scripts as settings (that merely modify parameters) and operations (that modify the view, structure, or components, like instantiating a new object). In contrast to other projects, we organize scripts – like all other courseware's components – in our database (3.3.3). We will describe later (4.1.2) how the courseware's generator applies templates and inserts referred scripts as JavaScript sequences into the final Web page. Our **script templates** work in a context-dependent manner: according to the target location, we insert scripts, for example, either as hyperlink and pictogram, or together with an illustration allowing the user to preview script effects. We designed the templates to



Eine weitere Verbesserung zur Modellierung von Kurven ist die **B-Spline-Kurve**. Sie ist durch Kontrollpunkte und deren konvexe Hülle geometrisch abschätzbar, besitzt also ähnlich intuitive Eigenschaften wie die **Bézier-Kurve**. Änderungen der Kurve sind lokal aus.

Die B-Spline-Kurve ist eine polynomielle Funktion auf einem **Knotenvektor**  $T = (t_0, \dots, t_n)$ . Die Kurve ist in Segmente mit jeweils  $m$  Kontrollpunkten unterteilt. Ein Knotenvektor  $T$  ist eingeschränkt durch  $t_0 < t_1 < \dots < t_m < t_{m+1}$ .

Lässt man schwach monoton steigende ( $\leq$  anstelle von  $<$ ) Knotenvektoren zu, so lässt sich die Stetigkeit auf  $C^{n-j}$ , wobei  $j$  die Zahl der Mehrfachknoten anzeigt, erhöhen.

Die **Basis** der B-Spline-Kurve ist eine Verallgemeinerung der **Bernstein-Polynome**. Bei fortschreitender Graderhöhung  $m$  konvergiert die Kurve zur Bézier-Kurve. Dies entspricht dem Knotenvektor  $T = (t_0, \dots, t_n)$ . Im Experiment: Bézier-Kurve hinzufügen lässt man den Grad konstant und fügt sukzessive innere Knoten ein, konvergiert die Kurve gleichmässig zur **Bézier-Spline-Kurve**.

**Figure 18: By embedding scripting instructions into hypermedia content, authors may illustrate the current topic directly within the user's learning object.**

avoid unwanted results of script combinations by restricting the script's scope to the current Web page. This also ensures that multiple instances of a learning object (embedded at different locations within a courseware) do not interfere.

Scripting in interactive courseware performs tasks such as illustrating content, synchronizing learning objects, visualizing alternative points of views, or introducing functionality incrementally. At first, educators can **illustrate** statements presented in text passages or slides directly within the corresponding interactive learning object (see Figure 18). Authors may for example design script settings to visualize specific setups or special cases, or readjust parameters to match the textual description. Script operations may exchange constraints and renderers, construct additional scene graph nodes, adapt the GUI, or even import self-defined components. Scripts respect the current state of a learning object, which means that none of the learner's modifications are lost in a session.

A second scripting application is **synchronization** of an object with others. Consider the common case of a guided tour; with each step, we naturally want to match a learning object's state with the described setup. With scripting, we can now synchronize them, step-by-step, and learners may start their self-studies at any point.

*Example (Scripting Applications) Consider the use of hyperlink scripting depicted in Figure 18. The current course text defines and explains basic properties of a B-Spline curve. A corresponding interactive learning object offers the learner to discover them constructively. The author*

motivates the curve by comparing it with a Lagrange interpolation. We enriched the hypertext with scripts modifying the interactive learning object to visualize both the B-Spline and Lagrange interpolation (Figure 18, left). The learner may compare the curves' behavior graphically by modifying control points, degree, etc. Later on, another text passage compares the spline with a Bézier curve (Figure 18, right) and offers similar scripts to rearrange the learning object.

We can further employ the same learning object at different places in the courseware for visualizing the **alternative points of view** on the theory. (Even for the simple theorem of Pythagoras there are about 400 different known proofs.) Reusing learning objects introduced earlier in the courseware minimizes the cognitive load, as it provides an already familiar environment.

*Example (Alternative points of view)* The learning object introduced in Figure 10 pinpoints relationships between the scalar product of vectors and trigonometric curves. We can interpret the scalar product as the scalar portion of the projection of one vector on the other one. In case of normalized vectors, we obtain just the cosine of the enclosed angle.

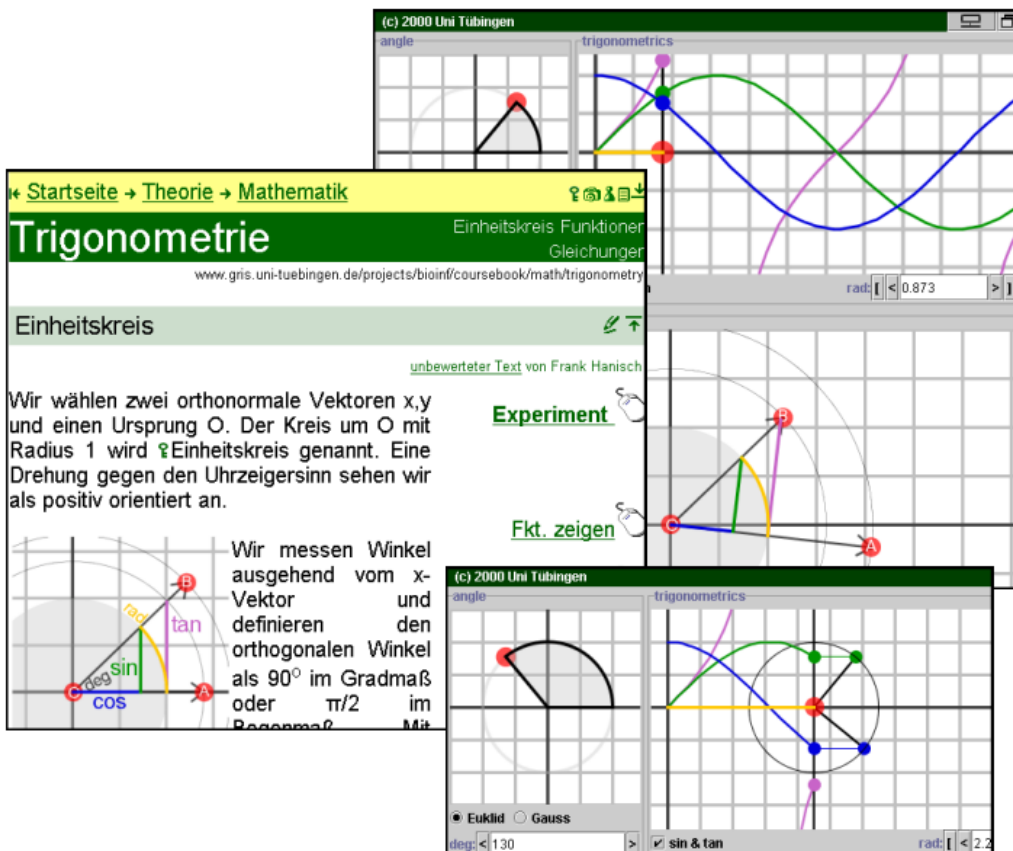


Figure 19: Scripting can visualize alternative points of view on the theory. This Web page (left window) revisits an already familiar interactive learning object (top window) to pinpoint a second interpretation; scripts rearrange the scene and data flow accordingly (bottom window).

*Vector normalization corresponds with a scaling to the standard circle; there, learners immediately understand the well-known relationships between sine, cosine and tangent by applying the theorem of Pythagoras and intercept theorems – rotating one of the vectors rotates the values around the standard circle and we obtain the trigonometric curves. Alternatively, another passage in the course text interprets the trigonometric curves as the locus of a fixed point on the rolling circle, and rearranges the scene and data flow accordingly. Figure 19 depicts the learning object’s setup before (top window) and after scripting (bottom window). Note that the data flow is bi-directional: learners can manipulate not only vectors, but also angle, curve points, and location of the rolling circle. We could explore more relationships by switching from Euclidean to Gaussian plain, e.g. the theorem of Euler.*

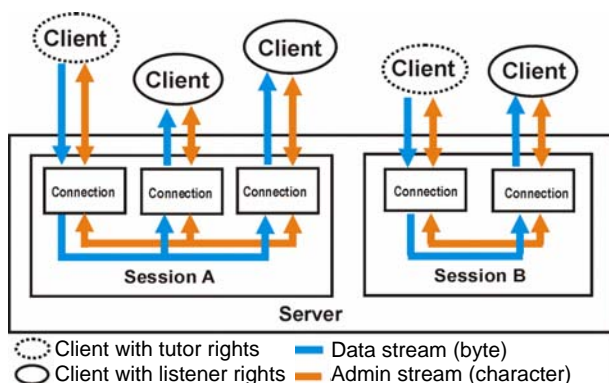
Lastly, transferring a learning object’s functionality into corresponding text passages or illustrations enables educators to **introduce functionality gradually**, and to reduce an otherwise overloaded GUI. We will give a detailed example later (5.3). Learners get only the functionality they need to understand the current text passage. Reading on (and activating corresponding hyperlinks), they meet other scripts that extend the learning object’s design, layout, or functionality.

### 3.3.2 Networking

To overcome a learner’s isolation in Web-based teaching, we generalized our scripting architecture (3.3.1) to a network model permitting collaborative work (2.3.1). We designed a simple client/server architecture that allows executing one online session per learning object, each of which can be used for either a classroom scenario or a consultation/examination. (We teach an average of 20 students per lecture, and do not have to perform any load balancing.)

Again, we equipped our base component for interactive learning objects (3.2.4) with necessary multi-user functionality. That way, any of our learning objects becomes network-compatible, a potential **client** to participate in an online session. (However, to be of value in a real-life scenario, a learning object’s setup must be well-developed.) The default layout now includes a networking button indicating the current state (inactive, passive, i.e. active but not participating in a session, and participating), and providing access to the learning object’s active user list and chat. Our communication model imitates the **classroom setting**. After logging in, a user may request to participate in a running session. If the tutor agrees, the user’s learning object registers as a net listener and becomes synchronized to the tutor’s learning object. A participant must not interfere, except that he may chat, or leave the session. However, the tutor’s role may be handed over, and the appointee may then demonstrate an action, or carry out some task. Apart from its use as a classroom scenario, we can also utilize the single-tutor-many-listeners (1:n) model for remote consultation/examination (n=1).

Users may enter a session at any time. To perform the required synchronization, we make use of scripting. The state of any scriptable



**Figure 20:** An extended scripting architecture allows networking in a classroom setting. The tutor transmits scripts and parameter changes to all participants. Control can be handed over.

arbitrary script operations modifying the object's view, structure, or components. Our current prototype simply sends all scripts ever made (disconnecting the participant's view during this period to avoid flickering). Note that such a **dynamic script generation** mechanism enables us to create script settings on the fly.

The **server** organizes the list of participants, and synchronizes the state of their learning objects. We set up one session per learning object identifier. The server uses two separate ports for administration and data (see Figure 20). As we restrict ourselves to one-way data communication, we only have to deliver scripts and parameter changes from the tutor to all participants. Furthermore, constraints update dependent parameters client-side. We therefore have to transmit initial parameter changes only, which substantially reduces the number of script instructions as well as the netload. Entire scripts are transferred only if they are undefined, otherwise we simply send their identifier.

*Example (Networking) Let us enable the learning object introduced above (see the previous Example, and Figure 19) for networking. Although too straightforward to be applied fruitfully in a real-life scenario, it still demonstrates the economy in data exchange. Server-side, we simply register its identifier to the session manager's list of valid learning objects. For synchronizing the clients' states, we have to transfer angle and vector point locations – all remaining parameters are hooked on them via constraints. Therefore, we declare both angle and vectors as net properties. If we have allowed users for scripting the global scene transformation (i.e. letting them zoom, rotate, or translate the scene), the corresponding transformation matrix must become a net property, too.*

For reasons of security and reliability, we permit only registered scripts to be performed in networking. In the context of community support (4.2.2), we will discuss how we could set up such a set of registered scripts automatically utilizing a rating system.

object can be queried; the object in turn generates a state-setter script that can be used in scripting to reset the object's state to the one at hand. A learning object, respectively its scripting component, must provide similar functionality. In general, we gather all script instances, combine their identifiers and state-setter scripts, and send the overall script sequence to the new participant. However, remember that the tutor may perform

### 3.3.3 Scripting Database

*“Scripting opens up significant new possibilities for interactive guides. [...] A remote person or computer guide could send scripts [...]. Guides could be supplied by an open, free market [Roschelle96].*

With reusable software components and scripting, we can now advance towards the organization of highly interactive learning objects in digital libraries. We already emphasized the need for decentralization and community support (2.3.3). Chapter 4 will discuss our Web framework concerning Web-based authoring, content management, and production. Here, we focus on aspects of our scripting database.

Developers cannot foresee and implement all the desired functionalities of a highly interactive learning object. A framework covering all fields in SMET education exceeds the manpower of any group of developers (2.4.1). Firstly, creating highly interactive learning objects requires expertise in the subject, programming, pedagogy, didactics, and design. Secondly, employing them in a Web-based courseware requires a team of educators, tutors, and administrators who are able to perform Web-based authoring and scripting. A **decentralization** of the required knowledge might be the only way to guarantee a courseware’s sustainability and continuous enhancement.

From the learner’s point of view, Web-based courseware must offer cooperative tools such as discussion boards, chat, or annotations. The feedback obtained from our students using such tools reveals the need for referencing a specific state of an interactive learning object, or sharing their own setups. Some of our programming exercises result in new scripts, software components, and learning objects. Of course, we would like to integrate such work into ours. Similarly, educators and other end users are likely to produce their own extensions. How may we benefit from the community’s work, that is, include user-defined scripts, components, or entire learning objects? Our answer is to deal with them as with all the other, non-interactive content. Collect them, organize their core data in a database, require metadata, generate final objects via templates and style sheets, share them with other users, make it possible to refer to them in editing, and let community members annotate, rate, and modify them.

Note that our interactive learning objects do not have only primitive properties (e.g. boolean/integer/string data type), but arbitrary scripts. While the script identifier must provide a constant, unique reference for hyperlink scripting (see the within-component layer of the *Dexter Hypertext Reference Model*, 2.4.3), script content may change over time. We respect the need for reliable anchor targets by setting up a set of registered scripts providing valid targets in editing (i.e. for inserting hyperlinks with scripting instructions in chat, discussion boards, course text, and other Web content).

The learning object’s **metadata** specifies identifier, title, category, classification, interactivity level (3.1), corresponding project, abstract, illustration, initial dimension, main Java class, initial script (performed on startup, 3.3.1), required component packages (JAR files), required software environment (Java plug-in, Java packages), authors, version, and a

timestamp. Script metadata consists of identifier, title, learning object identifier, type (setting/operation), status (unofficial/registered), abstract, and illustration. Likewise, script instances come with identifier, title, script identifier, class, and description.

Now, how does this approach fit in with current learning management systems (LMS), and in particularly with **learning technology standards** such as the Sharable Content Object Reference Model (SCORM, 2.3.4)? We have already stated that current specifications do not support highly interactive learning objects properly. However, we believe that future standards will include advanced issues of interactivity, components, and scripting. We therefore wish to outline the application of the SCORM to interactive learning objects with a scripting architecture, and illustrate some of our most urgent needs.

In SCORM terminology, course text, illustrations, tables, scripts, and interactive learning objects represent assets, arbitrary pieces of a sharable content object (SCO). One of SCORM's metadata categories defines relationships between assets or SCOs; for scripting, for example, we would use best practice vocabulary 'ispartof', 'requires', or 'references'. SCORM restricts the kinds of relationships to a predefined set of values. We would like to extend this set, that is, to include e.g. an 'iscounterpart' relation for script operations that have inverse scripts (undo functionality). Moreover, describing the learning resource type with best-practice vocabulary (simulation, experiment, problem statement, self-assessment, exercise, diagram, figure, graph, table, and narrative text) immediately turns out to be problematic if we consider dynamic, scriptable content – which enables end users to modify all aspects of the learning object. There is a similar problem in the case of metadata such as learning time; the SCORM mainly deals with static learning units that learners work through in one go. In contrast to that, we aim for an interlinking of (complementary or alternative) learning objects content that suits learning best in a cognitive sense.

SCOs can communicate with the LMS and store and retrieve string values (resolved through JavaScript calls, which fits our approach perfectly). We can use this mechanism to launch an interactive learning object in a specific state, or store its current state for later use (via dynamic script generation, 3.3.2). However, the model prohibits SCOs to set values of other SCOs (or ask the LMS to do so). Any means for synchronization, adaptation, or other scripting applications would be lost. As a workaround, we would have to bundle all assets interlinked by scripting into one, large SCO.

Another key problem is SCORM's current launch model: it allows only one SCO to be active at a time – but, we want learners to work simultaneously with our synchronized theory and interactive learning objects. Again, this would be a motive for creating bundles.

### 3.3.4 Drag & Drop Scripting

Let us finally come back to our notion of highly interactive learning objects (3.1). We have required direct manipulation in all aspects, including object view, all essential object parameters, and object functionality (structure, components). While GUI and scene graph components already meet major demands, we are still not satisfied with the symbolical meaning of hyperlink scripting, which simply hides programming aspects behind a hyperlink, and provides no means for cognitive association between script and target object.

Furthermore, browser plug-in technology seriously restrains interoperability between hypermedia objects (2.2.3, 2.4.3). Typically, the script's scope is restricted to the current browser document, or to its siblings. We can not script local learning objects from Web pages, and, vice versa, scripts located outside the browser application (e.g. within a word processor, or a presentation program) will not work with Web content. More generally, only hypermedia objects belonging to the same context may communicate through scripting.

We therefore introduce a visual scripting mechanism, **Drag & Drop Scripting**, which communicates parameters and functionality beyond the browser barrier – between other hypermedia objects or native applications. Scripting instructions are encrypted into standard images, and performed by physically operating the image on the learning object, or on parts of it. The basic idea of Drag & Drop Scripting is as follows: (1) source out a learning object's functionality into scripting operations, (2) encrypt a script in an image that illustrates the result of the script, (3) permit the user to drag and drop the image onto the learning object, and (4) decrypt the scripting operation inside the learning object and execute it.

Remember that Drag & Drop (DnD, 2.1.1) represents a platform-independent direct manipulation paradigm operating beyond application boundaries and that it is part of the user's familiar desktop environment. Nearly all of the Java Swing components support DnD natively, others have to implement a minimal DnD API [Sun98]. We transfer all scripting functionality to the DnD action's source and destination object; that way, our approach works even if the user has disabled browser scripting functionality.

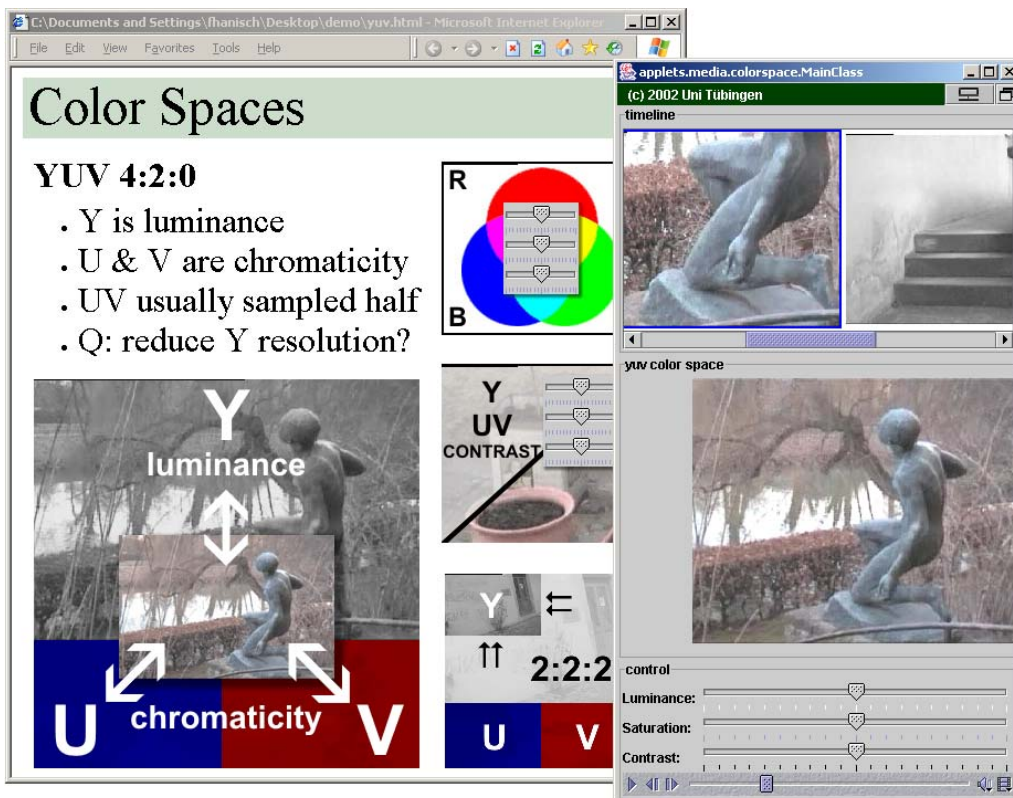
Our framework organizes scripting instructions in a script database (3.3.3) and steers Web page generation by templates (4.1.2). Illustrating images are part of our script metadata. The script template converts scripts into a corresponding HTML sequence, creates a thumbnail, and embeds it into the final Web page. In addition, it now hides the script in the image. Currently, we perform a least significant bit (LSB) insertion that works only with lossless image formats. In fact, each pixel stores 2 bits of our data. In the case of 8-bit images, which are less forgiving to LSB manipulation, we simply color the pixels in the Web page's background color. An improved version would use watermarking or a steganographic system [Johnson98], and support JPEG images. We start with a header (magic number, learning object identifier, border color, etc.) that enables



the learning object to identify script images, and validate or deny the drop action. The subsequent data block contains the script.

Next, the learning object has to become aware of DnD actions. Any of its GUI components that may receive a drop action must implement the DnD API and delegate work to our scripting component. In this manner, we have prepared fundamental components of our architecture, which already cover the bigger part of our learning objects for Image Processing and Video Processing (e.g. image browser, image viewer, video player).

*Example (Drag & Drop Scripting):* Figure 21 demonstrates an interactive learning object teaching basics of color spaces in Video Processing, respectively RGB and YUV color spaces. We have provided scripts and images that modify parameters of the video renderer (to visualize YUV channels separated, or combined), rearrange the object's layout (to insert controls for lightness/saturation/contrast or the amount of red/green/blue), and apply other YUV formats. Learners perform a script action by placing the corresponding image on the learning object. For example, a course slide (left side) might ask the learner about the effect of reducing not only color information (U and V), but also luminance (Y). By dragging the accompanying image (left window,



**Figure 21:** This interactive learning object illustrates color spaces in Video Processing. A Web page provides theory and scripts (embedded into images) that may be operated on the object via Drag & Drop. Users may drop video frames to the timeline or any other location, including native applications.



*bottom center) into the interactive learning object (right window), the learner can experience in a simulation that the eye is more sensitive to luminance detail than color detail.*

*When the user starts a dragging action in our video renderer, we extract the current video frame and embed a script sequence that will prompt the video player to reposition the video stream according to the frame's timestamp. The user can drag the frame to the object's timeline (right window, top panel) or any other application that supports DnD. Note that applications that resample DnD images (e.g. Microsoft PowerPoint) will distort a primitive LSB encryption.*



## 4 Web Framework

4.1 Organization and Production.....	84
4.1.1 Layered Database Model.....	84
4.1.2 Template-Driven Generator.....	85
4.1.3 Offline Management.....	87
4.2 Web-based Authoring .....	89
4.2.1 Online Wizards .....	90
4.2.2 Learner Support .....	92
4.2.3 Author Support.....	95

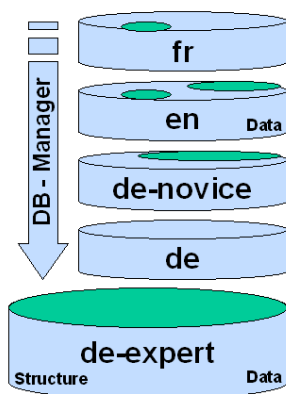
## 4.1 Organization and Production

Courseware, or learning management systems in general, must perform a sophisticated content management (2.3.3) to supply community members with means for content creation, modification, and extension. In this chapter, we describe in detail how we organize core data by a layered database model (4.1), generate all Web pages automatically (4.1.2), and offer an offline tool for managing content, structure, and design templates (4.1.3). The subsequent chapter will focus on online wizards (4.2.1) supporting learners (4.2.2) and authors (4.2.3)

### 4.1.1 Layered Database Model

Content management is based on database technology to provide efficient, large-scale query and update functionality. Since databases differ (for example in object type and maximal allowed string length), and data can be physically spread to many databases, we developed an abstract database manager. The manager interacts with the underlying databases and simplifies both queries and updates. The high-level, platform-independent Java database interface (JDBC) enables us to employ any standard relational database.

We support two types of structuring, a horizontal one separating structure, content, and design, and an orthogonal, hierarchical one using a set-based data model and metadata (2.3.3). Our proposed layered database model modifies the second structuring by introducing **layers** – data is distributed into layers according to its attributes (see Figure 22). Similar to the use of sets, layers enable us to organize alternative versions of content such as multilingual data or multiple depths in information. Each version corresponds with one layer; if a UNL (2.2.2) request fails for one specific layer, the database manager queries the next layer in the hierarchy.



**Figure 22: A layered database model supports alternative versions of data and incrementally given data while keeping links consistent.**

Layer-independent data and structure information has to be stored on only one layer.

A major benefit of such a model is **link consistency**. The database manager will automatically fill content gaps with an equivalent version from some other layer, which will eventually lead to mixed (but valid) content, like a Web page containing parts in different languages. This mechanism is carried out implicitly, i.e. developers do not have to bother with versions of data; they simply set a preferred layer, and perform their query, insert, or update in familiar SQL syntax. Database actions may target either a single layer, or all layers. While single-layer actions typically insert layer-independent data or update incrementally given data, actions targeting all layers insert or remove layer-dependent data such as course text. The manager assures that such actions are performed

consistently; for example, a single “insert subchapter” action leads to changes on all layers, as well as in structure.

Besides issues of internationalization and adaptation of the level of detail, another reason for us to introduce layers was the fact that it allows us to support **incrementally given data**. Authors may now supply their input without having to fill in data for all layers immediately. Our online wizards (4.2), which allow for Web-based authoring of the courseware, come with built-in layer support; they initially prompt the user to provide data for the default layer, and leave input data for all other layers optional. Requiring less data lowers the inhibition threshold for user-side extensions, and enables us to incorporate multiple authors in content creation, e.g. translation.

*Example (Layered Database Model) Figure 22 illustrates how we represent multilingual content by layers. We have registered three languages to the database manager, and defined corresponding successors. Authors may now concentrate on a single language (here, the German language) and include more translations (here, to English and French) step by step, which speeds up content creation. If our database manager queries specific data that has not been translated yet, it redirects the query to some other version. In addition, we subdivided the German version into an expert (short), a standard, and a novice (detailed) version, each represented by a layer. Language-independent data like structure, images or videos without text, and audio are stored only on the lowest, German expert layer.*

Although not stipulated by the model itself, we found out that both maintenance (backup, translation/adaptation) and export of a single version are cut down significantly if we use exactly one physical database per layer.

Using **metadata**, we achieve a more graduated structuring. Remember that we extended the LOM specification by custom metadata, e.g. for scripts and instances (3.3.3). Our courseware generators (4.1.2) apply specific metadata filters to create content versions. For example, we remove optional marginal information, lengthy examples, or annotations to create short versions; conversely, we insert them in detailed versions together with questions, gap-filling texts, and self-tests. We furnish data and metadata comprising a predefined set of values with **dictionaries**, which are employed by the database manager to translate one-layer data to other layers on the fly. Most obvious is the application of a dictionary with internationalized keywords (see case studies, 5.1); others, e.g. abbreviations, are possible. Note that dictionaries work context-sensitively: a specific translation in the context of one object type might differ from translations in other contexts. Therefore, any of our dictionary entries requires the specification of an object type.

#### 4.1.2 Template-Driven Generator

Our hybrid generators balance static and dynamic content (2.3.3) and automate all courseware production. Now, we briefly outline static content

production; the dynamic part will be discussed in the context of online wizards (4.2).

Let us come back to our horizontal structuring separating data into content, structure, and design. All of our generator components must combine these parts to produce final Web pages. Data consists of different types of hypermedia elements like course text, links, illustrations, multiple-choice tests, interactive learning objects, and scripts. We represent only primitive structure information (object and courseware hierarchy such as parent/child/siblings relations, and sets defined by metadata) as database objects, and transfer matters of navigation and design into **templates**. Our templates can be defined textually, or be hard-coded in Java to perform complex data operations and image processing. We created a template parser to enrich standard HTML/XML blocks with algorithmic functionality (variables, if-clauses, and loops), database queries (SQL syntax, or a simplified data iterator), and common image/file operations (e.g. thumbnails, watermarks, file transfer between database and server). Templates are vital during the courseware's warm-up or evaluation phase – they enable us to customize design, layout, and content filters quickly by performing little changes to the corresponding templates. Generator components feed templates with context-dependent data (current layer, path to root, parent, children, siblings, referring objects, relative location in document, etc.) and thereby provide a mechanism to define context-dependent template blocks.

*Example (Design Templates) Our design template registered for illustrations imports an image from the database, converts it to the specified dimension and image type, and overlays a watermark. We have defined it context-dependently. Consider therefore a layout pattern with main text and marginalia: if an author places an illustration within the main text, the template displays it directly at that position, letting the current paragraph flow around it. If the illustration is placed in the marginalia, a thumbnail with an hyperlink to an additional page is created, which displays the image and its subtitle together with references back to all objects containing the illustration. In the first case, we display a given subtitle as mouse-over text. The template is given textually - modifying the design is trouble-free and requires no deeper programming knowledge or tool.*

We update our courseware daily (see case studies, 5.1); on demand, we initiate the **generator phase** immediately. A generator requests a courseware's hierarchy structure, fetches the root node, recursively collects its children, and traverses them. We interpret a set of leaves sharing the same parent as belonging to a single document. The generator processes the document template, and composes a Web page accordingly. Each leaf may contain arbitrary hypermedia objects; they are embedded into the final Web page using the templates registered for that kind of objects. All courseware, including course text, illustrations, embedded interactive learning objects, and scripts, and several lists and indices (table of contents, figures, glossary, interactive learning objects, bibliography, member list, etc.) is produced in this way. We further include content belonging to the dynamic part of the courseware that is rated to be useful

as static content (annotations, frequently asked questions, ratings, etc.). Finally, we provide a mechanism to exclude complete directories from the generation phase, and create (or reference) a default Web page only. This proves practical for self-contained subprojects that usually want to design a custom Web page. Similar, we do not generate the courseware's root page, as it is typically designed individually (e.g. a welcome page or overview).

A link database (2.2.2) organizes links as separate UNL objects to ensure link consistency. Links are typed, and generators may process them bidirectionally, which lays ground for an adequate courseware **interlinking**. In general, interlinking refers to (1) connections between entire hypermedia objects, (2) connections within an object, and (3) connections between an object's within-component layer and other hypermedia objects. Interlinking object parts becomes complicated when dealing with pieces of software; we have therefore dealt with software components and scripting (3.2.3, 3.3.1). Creating bidirectional connections between entire hypermedia objects, interactive or not, can be accomplished by templates. Whenever we encounter a link object during courseware generation, we retrieve the corresponding link template for that type, process the block matching the current context, and embed object references into the Web page. To insert back-references, we follow one of the following strategies dependent on the link type and context. For 1:1 connections, we simply insert the back-reference when we process the target object. Otherwise, we may immediately insert all existing back-references, either at the place or in the target document (see Example above). As core data may be reused in multiple coursewares, we restrict back-references to the current courseware by introducing a courseware identifier. Lastly, we may delegate back-referencing to some other template.

*Example (Back-References) To interlink our BibTeX bibliography (see case studies, 5.1.3) with community members adequately, we employ link templates that delegate back-referencing to some other template. We initially create all references from members in their bibliography entries, which we include as personal publication lists (template A). Finally, we process all objects containing lists of bibliography entries, and reference each entry back to its counterpart on the member's personal publication list (template B). Template A therefore stores corresponding linking information in template B.*

### 4.1.3 Offline Management

Our courseware management tool enables us to modify structure, content, and design offline. This tool is meant only to perform fundamental changes; most community members prefer Web-based authoring (4.2).

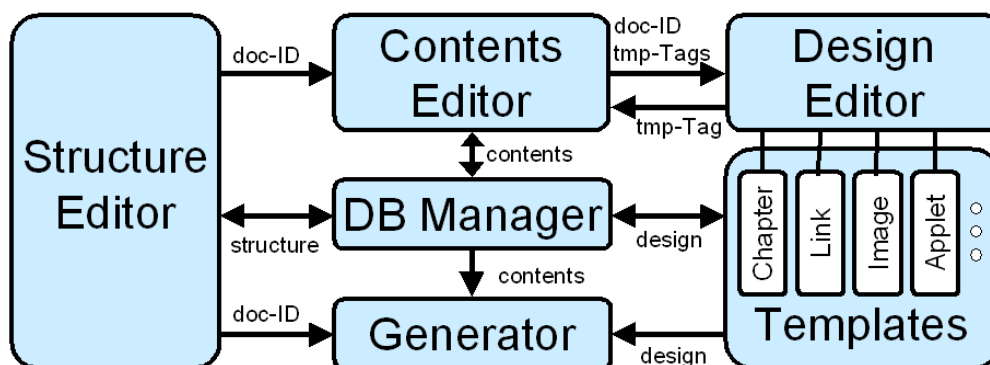
The tool's architecture consequently maps the horizontal data structuring to three **components**, a structure editor, a content editor, and a design editor (see Figure 23), which we implemented respectively as a tree editor, a text editor, and a property editor (see Figure 24). While the underlying database manager handles all data queries and updates, generator components create the courseware on request (4.1.2). Components

communicate standard JavaBeans properties; therefore, we can exchange them with any other text editor wrapped in a Java container. Most of our generator components produce hypertext, i.e. Web pages. For specific sub-structures like our interlinked PDF bibliography (5.1.3), we further implemented PDF generators. Slides and lecture notes could be integrated similarly. We can plug-in arbitrary template components; our courseware for example typically include templates for document, illustration, link, exercise, bibliography, multiple-choice, gap-filling test, interactive learning object, and script.

The actual data loading and storage is delegated to the database manager. Functionality and data flow is as follows (see Figure 23): on loading, the **structure** editor requests the courseware hierarchy and builds a matching tree. The author may rearrange courseware structure by drag & drop or cut & paste, and rename specific nodes. We enforce naming conventions to result in transparent path names (URLs). For instance, leaves forming document parts (4.1.2) automatically receive a link shortcut, which is used, for example, in the navigation block of the document template. As we built the link shortcut from parts of the full title using a set of plausible rules, the full title will naturally anticipate its shortcut. Once the author selects a specific node, we message the node identifier to the content editor, and the current generator component.

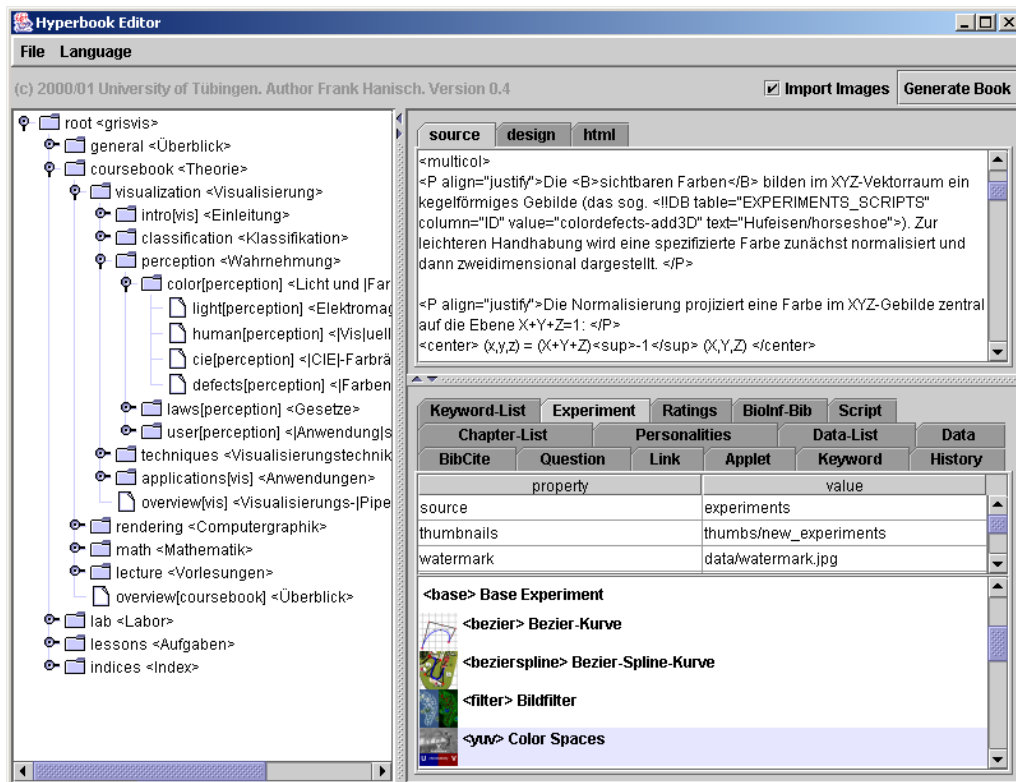
The **content** editor in turn queries the database manager to retrieve the actual content, and offers modification thereof. At present, we have implemented a plain text mode, a prototype design mode (providing icons for common objects, and simplified block actions), and a WYSIWYG preview. The editor constantly scans the input for object references (or “template tags”) that identify template type, object, and optional parameters, and compiles them to a tag list. Changes to the tag list are messaged to the design editor.

The **design** editor scans the template tag list and creates (or updates) corresponding template components. For each template, we generate an editable property list representing all template text properties, and include



**Figure 23:** We developed a courseware management tool for modifying our courseware offline. The component architecture reflects a strict separation of structure, content, and design. Data flow is denoted with arrows.





**Figure 24:** Our courseware management tool enables us to modify structure, content, and design offline. We may edit the courseware hierarchy in tree view (left side), edit a node's content in plain text or design mode with WYSIWIG preview (top right), and steer object embedding by template properties (bottom right). Activating a single button starts courseware generation.

any custom template GUI. Templates may query the database manager for available objects, and enable users to select and modify them, e.g. via a custom thumbnail list (see Figure 24). The design editor allows the user to switch the template type and access the template GUI. Selecting an object produces an appropriate template tag, which we send back to the content editor, and embed it at the current cursor position.

## 4.2 Web-based Authoring

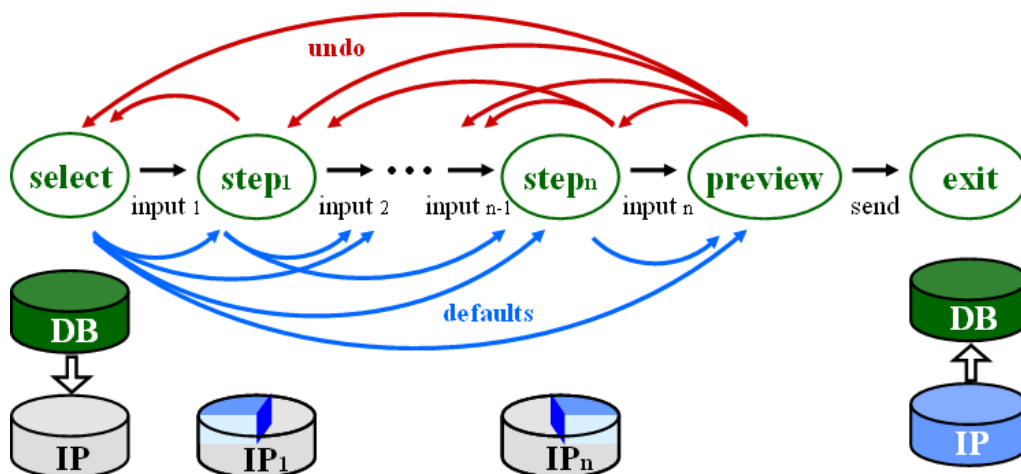
Organizing courseware objects in database layers (4.1.1) and steering courseware generation by templates (4.1.2) creates a basis not only for matters of maintenance and interlinking, but also for personalization and collaborative work (2.3.3). In the following, we present an appropriate state machine for online wizards (4.1.1) providing all community members with Web-based authoring functionality. An abstract base wizard allows us to derive new online tools quickly. We discuss some of the most essential wizards of our courseware, divided into wizards for learners (4.2.2) and wizards for authors (4.2.3). Keep in mind that our approach presumes a scripting database (3.3.3) to include fine-grained, interactive content.

### 4.2.1 Online Wizards

Our online wizards enable community members to annotate, modify, and rate arbitrary courseware objects, including scripts. For each type of object, we register an appropriate online wizard. Each object template (4.1.2) defines an activation block for its wizard accordingly, which can then be inserted into a Web page. Community members can activate wizards directly on site. To reduce the cognitive load presented to the standard user, we create Web pages both for study mode (offering only learning-related wizards, e.g. annotations, exercises, and rating) and for editing mode (containing all wizards); a single button lets the user toggle them. Technically, we create these versions by running the generator twice with different metadata.

The technological platform of our online wizards are Java *Servlets* (server-side applets). Servlets provide a component-based, platform-independent method for building Web-based applications, without the performance limitations of CGI programs (2.2.3). In particular, while CGI requests are answered in a separate process by a separate instance of a CGI program, servlets are loaded and initiated once, and stay in memory between requests. Moreover, servlets may answer requests concurrently.

We base our online wizards upon a well-defined **state machine** that comfortably manages authorization, default values, undo facility, preview, and problems like temporary log-offs, deactivated cookies, or firewalls (see Figure 25). All wizards start with an authorization step presenting an ordinary login screen to the user. Optionally, we identify a user by the Internet Protocol (IP) number (and keep him logged in forever; otherwise, he is logged out after some idle time). After successful authorization, the user may select his object of choice. Applications may skip this step and provide a default selection instead. We guide the user through all required data steps. During each state transition, we store the given data in a temporary IP database; data is reloaded from the courseware's database



**Figure 25:** The core of our online wizards constitutes a state machine offering undo facility and default values. Database actions are displayed below the corresponding steps.

only in the selection phase. The IP-based storage mechanism preserves data session-independently and works even if the user logs out during the input phase, or turns off browser cookies. After providing visual feedback in a preview phase, we send the given data from the IP database back to the courseware's database. We also send an e-mail notification to the editorial board in order to verify the data.

In general, the user may arbitrary step forward (we provide default values as far as possible by filling in existing data, or reusing previously entered data) and backward (undo) while entering data. Thus, we neither direct the user into dead ends, nor prompt for data twice. In case of obligatory or invalid data, we restrict forward movement and redirect the user to the corresponding step. Each wizard contains a set of rules specifying constraints and possible redirection (to a previous step, or an auxiliary step indicating the error). Rules that do not specify a redirection will stall step transition and set an appropriate flag to indicate invalid data.

*Example (Online Wizard): Community members of our courseware may modify personal member information by using of a corresponding online wizard. They activate the wizard through a hyperlink either on the courseware's member list, or on a specific member page. In the first case, we let the user select a specific member from a list (users need administrative rights to change other member's information). Several data input steps prompt for the member's name, affiliation, photograph, and other information. We provide default values for any data, except the obligatory name, which is required to enter preview. We check for flags indicating invalid name/image information; in case of errors, we redirect the user to the respective step, and highlight the input field appropriately.*

Our abstract base wizard implements basic functionality for user management (e.g. passwords, roles, groups), states (e.g. rules, variables, defaults), data operations (e.g. image/file upload, IP database management – we delegate all work to the database manager, 4.1.1), step design (uniform navigation and layout), internationalization (e.g. languages, dictionary), and e-mail notification. Any task-specific wizard is derived from this base component and ships with textually defined wizard properties and state **templates**. While the properties contain rules and multilingual text, the templates define the HTML block displayed in a given state (see the courseware generator's template parser, which uses the same mechanism, 4.1.2). Instead of defining states by templates, we may alternatively hard-code them. Note that a derived wizard inherits all functionality from its super component, including templates. That way, we designed the login screen only once for the base wizard; the same goes for rules like the obligatory username and dictionary entries (translations and abbreviations) for salutation, terms of copyright, and common expressions.

As we employ simple HTML forms for data input, we end up with plain text **editing** functionality only. In our current courseware, authors do not complain, as their content creation tools allow for HTML export. A more adequate Web form would implement a word processor (e.g. Macromedia Contribute) or use JavaScript and browser-specific functionality.

Community members may include references to other object types (course text, illustrations, glossary, bibliography, member pages, interactive learning objects, scripts, etc.) using vocative hyperlinks – textual pointers that match a given set of rules, like “see chapter nn” [Maurer96]. Wizard properties hold a list of search patterns together with phrases that may occur nearby. Optionally, members may define URN references explicitly using conventional XML syntax.

We spent more effort in creating an adequate GUI for editing multi-layered data. Consider interfaces that allow for modifying layered data independently: authors would soon become tired of changing between layers and produce inconsistent content. Therefore, we support parallel editing on multiple layers, e.g. editing of multi-lingual data or data with different depths of information. Our state machine contains functionality to iterate over all registered layers, and offers authors to switch layers at any state. This mechanism works also for layer-independent data. Normally, a wizard starts prompting for data in the language matching the user settings, and then iterates over all other layers in a subsequent step (reusing the initial input as default for all other layers). In specific cases, users may create alternative versions in a single step. We provide, for example, a simultaneous editing of short, standard, and detailed versions using a Wiki-like (<http://c2.com/cgi/wiki>) smart syntax, which simply marks specific blocks with metadata. The generator’s metadata filters will then decide if a block becomes visible or not. Of course, the smart syntax approach requires authors to set up content carefully.

Usually, the wizard GUI will adapt to the current layer, too. Consider for example multi-lingual data, which requires us to provide international data in- and output facilities. Our approach enables the user to change the wizard’s language settings in any state. We outsourced all textual information into wizard properties and provided translations for all registered languages. The base wizard fills the template according to the user’s language settings. Note that we translate not only the wizard GUI, but also the given data; for instance, if the user has given data in English and subsequently switches to a German layer, we check the dictionary for a matching entry, and, if found, display the German translation instead. Common phrases are translated automatically using a context-sensitive dictionary (4.1.1).

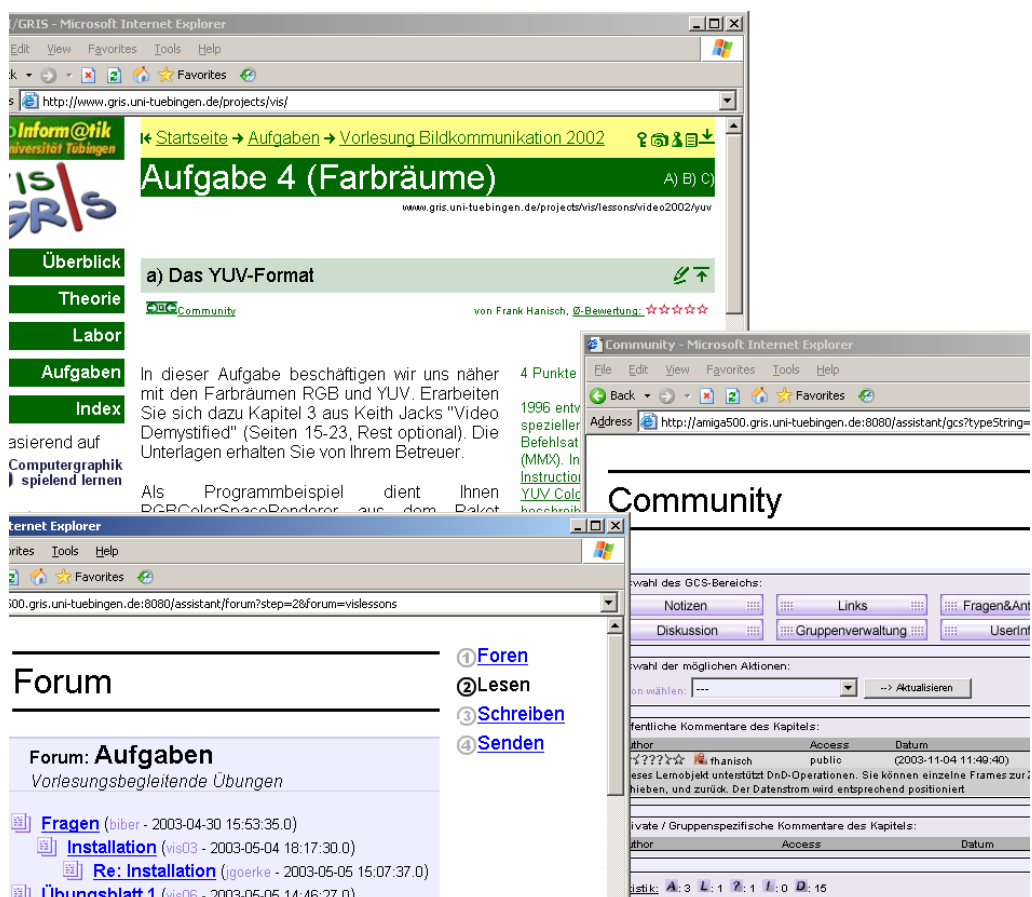
### 4.2.2 Learner Support

From the learners’ point of view, a courseware must allow annotations, structuring, and active participation in courseware development (2.3.1). In essence, we must overcome their a priori passive and isolated role by, for example, offering them to ask questions at any place, or to talk with other community members.

We differentiate community members by an extensible set of roles, e.g. administrator, owner of a contribution, active or passive author (an active author will react on learners’ questions), learner, tutor, lecturer, guest, anonymous, etc. Roles affect the set of possible actions a wizard offers. While a tutor or lecturer for example may execute a remote consultation, a

learner may only participate. Members may set up their own groups, and restrict access to their contributions to some specific group or member. Similarly, administrators may restrict wizards to specific roles, groups, or members. One of our wizards summarizes the **member status** together with a list of contributions (sorted by the courseware hierarchy), and enables users to change personal information such as name, affiliation, photograph, etc. Of course, only administrators may change a member's role. The wizard further offers to set notification flags for some substructure in the courseware hierarchy. If an incoming contribution matches the chosen flags (e.g. annotation, or discussion), the member gets a corresponding e-mail notification.

Our most basic online wizards handle annotations, context-help, a discussion board, and a rating system. We may attach them to arbitrary content types by registering them to a specific pair of object type and object identifier (see Figure 26). That way, contributions become object-centered instead of tool-centered, an important property regarding information structuring (cp. with document-centered vs. application-centered, 2.2.2).



**Figure 26:** Community support in our courseware renders any object as a potential target for annotations, context help, a discussion board, and our rating system. Here, we let community members discuss (bottom left window) programming exercises (top left window), and add their own notes, questions, answers, and links (right window).

*Example (Object-Centered Wizards)* All our interactive learning objects automatically provide a discussion board (see below) responsible for that specific object, or topic. We therefore equipped the interactive learning object's base component with functionality to switch to the discussion board directly from the chat. That way, users may discuss topics presented in the object in a separate area, without having to deal with discussion threads belonging to other topics, respectively learning objects. Similarly, our course text templates contain a block mapping course chapters to corresponding online wizards. Community members may annotate that specific course text, obtain help for its topics, or rate its quality directly on site.

Let us consider these wizards in more detail. Firstly, an **annotation** wizard offers object add-ons such as notes, questions and answers, and related links. This enables learners to personalize content and to work with the courseware actively. Furthermore, public contributions might help other readers to understand the learning content, and serve authors as starting point for improvements or corrections. Combined with the rating system, the list of questions and answers further implements a Frequently Asked Questions (FAQ) list. A similar wizard provides **context help**. We follow the same mechanism as for graphical scenes (3.2.3): again, we initially check for any available user selection (technically, we retrieve selected text with browser scripting). We collect hierarchically, bottom-up, entries for the selected word (or object), then for the current paragraph, and finally for the surrounding course text. Note that we employ context-sensitive entries and a rating system; otherwise, this approach would not scale for a larger number of entries.

Our **discussion board** wizard in turn provides means for asynchronous communication. We have designed it to be clearly structured, using topic trees. Boards and their respective sub-trees can be restricted to specific members or groups; in that case, they are not visible for others. Besides targeting a specific learning object (see the Example above), we may explicitly dedicate a discussion board to a particular subject. Each board notifies a list of moderators about new contributions, and offers optional e-mail notification for future replies to a given contribution. Of course, we highlight unread contributions per member. In our university setting, we favor – wherever possible – asynchronous communication; for instance, most of our students work out their programming projects in groups in our department's computer pool. We execute the bigger part of lecturer/tutor consultation in restricted discussion boards, and reduce additional labor by publishing individual threads for public benefit. (We have also implemented a HTTP-based chat wizard making synchronous communication possible. However, we did not observe a single reasonable session carried out in any of our projects).

To improve the quality of contributions, we employ a **rating system**. A wizard enables members to rate objects (course text, illustrations, interactive learning objects, scripts, etc.) on the fly, or, if they are willing to spend the time, hand in detailed reviews. Based on such evaluations, we set up profiles for authors, participants, and content, which, in the long-term, help us to improve our courseware. We grade authors and create

rankings of the best contributions, busiest authors, and most wanted supplementations or corrections. As an average, we simply take the median.

*Example (Rating System)* We apply our rating system to the list of questions annotated to some course texts. If any of them is rated as important (on the average), we notify the editorial board. For those, our system might already serve as a preliminary stage to finding the answer: unofficial readers' answers that are rated to be useful might serve as a starting point, whereas other answers might indicate weak content that evokes misinterpretations, or lack of motivation.

Finally, a learner may activate a **self-test** wizard to review learning content. We attached the wizard to specific sub trees of the course hierarchy. That way, a learner may infer subject matter accurately. In the enquiry phase, we collect all questions belonging to the given sub tree, and randomly choose some of them. We support multiple-choice questions with alternative verbalization; each entry consists of a set of questions and incorrect and correct answers; we pick each set randomly. If the learner's response contains incorrect answers, we reformulate the questions by picking another verbalization. Note that we make all statistics anonymous to protect the learner's privacy.

### 4.2.3 Author Support

Our base wizard component (state machine and state templates, 4.2.1) enables us to create wizards quickly. We have developed more than 30 **task-specific wizards** providing Web-based authoring facilities regarding matters of institution (members, projects, areas of research, etc.), bibliography (BibTeX entries, search, etc.), and courseware (course text, illustration, glossary, history/timeline, self-test, remote consultation, etc.). Our cases studies will present some of them in more detail (5.1 and 5.3). For now, let us focus on the most innovative ones, namely our wizards for interactive learning objects and scripting.

We have implemented wizard prototypes offering Web-based authoring of an interactive learning object's data, scripts, and documentation of created script instances (see Figure 27). We assist authors in specifying an **interactive learning object** in all data/metadata input steps, just as we would handle non-interactive content. In particular, we request required component packages, data and metadata, e.g. title, classification, abstract, initial script, main class, and illustrative image (see scripting database, 3.3.3). We supply default values. For example, if the user does not specify a main class, we use the learning object's base component. We require authors to test the initial script immediately at the preview step. A supplementary test page therefore presents the uploaded learning object (or its base component, i.e. an empty container) together with a scripting text area containing the initial script. The author may execute the script to ensure correct behavior. Moreover, the author may change the script, and review the outcome immediately.

Two other online wizards deal with the definition or modification of a learning object's **scripts**, and, afterwards, with the documentation of



created script instances. Again, authors have to test their scripts at the preview step. If a script defines new instances, we prompt for the instances' title, class, and description. We detect a default class automatically. The class will be used later to interlink a learning object's help section with the API section of the programming guide.

Multiple authors may now create and modify interactive learning objects and accessory scripts just with browser functionality. Even untrained authors can perform scripting if guided by examples. However, scripts created by different authors may have poor **compatibility**. To avoid major problems, we restrict custom defined scripts to work with the initial set of packages only. Authors will eventually have to create a new learning object based on an extended set of packages.

Some scripts will require changes of existing scripts. We reduce scripting errors through compatibility checks that match modified script instructions with a set of pre-defined script patterns [Hanisch02a]. For example, we tolerate the conditional call of a ,setXXX' script, if the condition tests the existence of an instance defined by another ,doXXX' script. This assures that script behavior does not depend on effects of the ,doXXX' script. Modifications that go beyond this can be suggested in the learning object's discussion board, but are likely to be performed only if there is no doubt that there will not be any negative side effects. Our rating system will detect other obstacles: after a warm-up phase, we can identify high quality learning objects, software components, and scripts, or deprecate other ones.

Remember further that we generalized our scripting architecture to a network model, usable in a classroom scenario, or remote consultation/examination (3.3.2). A server delivers scripts and initial parameter changes to all participants. In offline mode, users may test

**Figure 27: Online wizards for interactive learning objects guide authors through authorization, informational data (multilingual), metadata, scripting, and preview.**



arbitrary scripts; but in networking, we allow tutors to perform only **registered scripts** verified as useful, bug-free, and secure. Our technical experts base their decision on the corresponding results of the rating system. The main reason for introducing registered scripts was reliability – tutors should never be allowed to render a learning object unstable. As scripting is resolved client-side and learning objects might, for example, have permission to access the file system, a learning object achieves the ‘secure’ grade only through staff ratings.



## 5 Case Studies

5.1 Electronic Webmaster .....	100
5.1.1 Project .....	100
5.1.2 Institution .....	100
5.1.3 Bibliography .....	103
5.2 Image Processing and Video Communications .....	105
5.2.1 Project .....	105
5.2.2 Visual Programming .....	105
5.2.3 Component Programming .....	108
5.3 Scientific Visualization.....	110
5.3.1 Project .....	110
5.3.2 Scripting.....	110
5.3.3 Community .....	114

## 5.1 Electronic Webmaster

### 5.1.1 Project

Let us start the case studies with a side project demonstrating the capabilities of our Web framework, in particular data layers (4.1.1), generator templates (4.1.2), and online wizards (4.2.1). The goal of the project was to create an *Electronic Webmaster* for maintaining our department's homepage (<http://www.gris.uni-tuebingen.de>), i.e. to provide both content management and Web-based authoring. The homepage should list current and senior staff, areas of research, projects, a gallery, vacancies, events (conferences, workshops, etc.), and publications. Web pages should become multi-lingual; besides German, English, and French, we had to include the local dialect Swabian to celebrate the 50<sup>th</sup> anniversary of our region, Baden-Württemberg in southwest Germany, in 2002. Content should be maintained consistently and become fully interlinked. Web pages should be generated automatically, including member pages with personal publications and links to related material. Finally, the core data had to be made accessible for other applications. Bibliography data, for example, should be reused in the department's annual report.

### 5.1.2 Institution

The department Graphical-Interactive Systems at the Wilhelm Schickard Institute (WSI/GRIS) was founded in October 1986 by the appointment of *Wolfgang Straßer* at the University of Tübingen. Today, about 25 research assistants and Ph.D. students work in areas related to interactive computer graphics. Our area of research for example, Interactive Web-Based Courseware, has produced coursewares accompanying lectures in Computer Graphics (a two-semester, four hours per week course taught by *Reinhard Klein* and awarded with a 'Landeslehrpreis'), Geometric Modeling, Computational Geometry, Image Processing, Video Communications, and Scientific Visualization. The next two case studies will describe the most recent courses in more detail (5.2, 5.3).

Using our Web framework consisting of a layered-database model, an abstract database manager, a template-driven generator, and online wizards with a built-in state machine, the Electronic Webmaster's implementation phase was straightforward. We had to arrange multi-lingual layers, define the department-specific look by designing templates, and create task-dependent online wizards allowing for Web-based authoring. As we could import most of the core data from other sources, only a single type of data, the BibTeX bibliography, took some effort. We will therefore describe the bibliography part in its own section (5.1.3).

We started with setting up database **layers** for the desired languages German, English, French, and Swabian. We registered corresponding ODBC data sources, which can be accessed using a bridge to JDBC. Our database manager may connect to any standard JDBC/ODBC data source; in our case, we simply based it on Microsoft Access, as that was already

contained in our software environment. After taking an inventory of the department's active data, we set up object types, (respective database tables) for chapter structure, chapter text, staff, illustrations, links, seminars, areas of research, projects, vacancies, events, gallery, bibliography, hardware reservation, and a discussion board. While we had to enter for example project data on all layers (multi-lingual title, keywords, and abstract), we could organize staff data (title, name, profession, room, e-mail, homepage, phone, member type – assistant, student, senior, etc.) on a single layer, and offer corresponding dictionary entries for a member's profession.

Next, we designed appropriate generator **templates** for each object type, defining the department-specific look. Object types such as structure, chapter text, illustration, and vacancy, which contain rather primitive elements (HTML blocks, variables, and lists), could be handled easily by text templates. In cases like staff, area of research, or seminar, which require nested iterations and the creation of additional documents, we built text templates covering parts of the final Web pages, and combined them by low-level programming. Consider for example a staff member's personal page: we iterate over all members, collect each member's general data, areas of research, projects, seminars, and bibliography, retrieve template blocks respectively, and combine them into the final personal Web page. We generate several versions of the publication list, including plain text style, BibTeX style, abstracts, and PDF. Both HTML and PDF lists include links to the full papers and supplementary material. Lastly, our JPEG generator supplies a printable business card.

Our generators rebuild the department's homepage daily. As noted earlier (4.1.2), some Web pages are excluded from this generation phase. For example, we do not create projects pages; instead, we generate a project list with title, members, and keywords, and link to the project's default Web page (i.e. we delegate management of the project subtree to project members). Second, we create area pages only in part; while we generate the area list normally, we defined a specific area page's overall layout by templates. The templates automatically insert sections for navigation, associated projects, staff members working in that area, and related publications. Afterwards, we mark the content section with tags and let the members fill in the blank space to suit their needs. Authors indicate changes by triggering an online wizard, which in turn transfers content into our database.

Other online wizards offer Web-based authoring of institutional data. Except structure, which we entrusted to the Webmaster only (using the offline management tool, 4.1.3), staff members may modify objects of any type. All **task-dependent wizards**, from chapter text, staff, illustrations, links, seminars, areas of research, projects, vacancies, events, and gallery to, finally, hardware reservation, use the standard data flow of the state machine. As a result, we were able to realize all states with text templates. Step one typically offers selection of an existent object or the creation of a new entry, step two queries general data in the default language (for most of our staff members, this is German), then requests non-textual data (images, video, PDF, etc.), and prompts for internationalized data (English,

---

# Autorenbeitrag

---



① [Start](#)

② [Schreiben](#)

③ [i18](#)

④ [Senden](#)

Bitte nennen Sie uns ausserdem den **internationale** Inhalt des Kapitels.

<b>Titel:</b>	<input type="text" value="Our Department At A [Glance]"/>	 <b>Englisch</b>
<b>Contents:</b>	<pre>&lt;p align="justify"&gt; The department Graphical-Interactive Systems at the Wilhelm Schickard Institute for Computer Science (WSI/GRIS) (WSI/GRIS) wurde im Oktober 1986 mit der Berufung von &lt;!!DB table="people" column="name" value="Wolfgang Stra�er"&gt;</pre>	
<b>Titel:</b>	<input type="text" value=" Pr�sentation  du Departement"/>	 <b>Franz�sisch</b>
<b>Contenue:</b>	<pre>&lt;p align="justify"&gt; Le d�partement des syst�mes graphiques interactifs de l'informatique fut fond� en octobre 1986 avec la nomination de &lt;!!DB table="people" column="name" value="Wolfgang Stra�er"&gt; � &lt;!!DB table="links" column="link"&gt;</pre>	
<b>Titel:</b>	<input type="text" value="Onser  Lehrschtuhl "/>	 <b>Schw�bisch</b>
<b>Inhalt:</b>	<pre>&lt;p align="justify"&gt; Dia Leit von de Graphisch-Interaktive System am Wilhelm-Schickard-Inschtitut f�r Computerles (WSI/GRIS)</pre>	

**Figure 28:** To celebrate our region's 50<sup>th</sup> anniversary, we created a multi-lingual homepage for our department. Online wizards request data in German, English, French, and the local dialect Swabian.

French, and Swabian, see Figure 28) simultaneously in a fourth step. After preview, we notify the Webmaster via e-mail.

The templates realize an adequate interlinking of all objects. A sample walkthrough might start at the homepage's welcome page, move on to the staff list, retrieve a specific member, look up a paper on his publication list, follow a link to one of the member's projects, and then to the area of research the project belongs to, and from there, on to other members working in that area, or to their publications. We decided on relevant paths (and interlinking) by evaluating Web server statistics (see Figure 29).

Wizard properties define obligatory data (e.g. area title) and provide context-sensitive dictionaries (e.g. member profession). To facilitate user input of timeframes (e.g. conferences and workshops, or hardware reservation), we implemented a string-based date parser supporting multi-lingual input. Note that our wizards adapt both GUI and given data

automatically, whenever an author changes language settings. Switching from German to English for example will cause a wizard to prompt for English data in step two, and include already given German data in step four.

### 5.1.3 Bibliography

Community support for bibliography data entails facilities for uploading data, generating lists, searching the data, and Web-based authoring. Again, we employed a generator component and state templates; furthermore, we had to implement a *BibTeX* parser supporting the common LaTeX data format.

Firstly, several **upload** wizards allow the inclusion of data entries in common bibliography formats (BibTeX, HTML forms, free text) together with full papers (PDF, PostScript) and supplementary material (video, slides). Data is stored in BibTeX format. We handle bibliography data like all other data; appropriate templates steer the filtering and generation of publication lists belonging either to individual staff members or to all staff members and some timeframe. Publications belonging to our department are clearly marked by a specific BibTeX keyword.

A second class of bibliography wizards realize a dynamic **search** (2.3.3); either a quick search, a professional search, or a LaTeX-specific citation search. While the quick search retrieves bibliography entries containing a given search text, the professional version lets the user precisely define pairs of BibTeX fields and search text. The citation search retrieves entries matching a given AUX (auxiliary) file that records citations in LaTeX. The wizards can also be queried by other applications. That way, we create the bibliography section of our department's annual report on the fly: staff members just have to cite the bibliography keys of our bibliography database (each entry includes a set of aliases), and let LaTeX create the AUX file, which is then utilized to generate the LaTeX BIB file automatically.

We further adapted the state machine's data flow to permit modification of existing bibliography entries. Authors first retrieve existing data using one of our search wizards, then **edit** the BibTeX entry in a HTML form, and submit it again to the upload wizard. Duplicate entries using the same bibliography key require authors to synchronize inconsistent fields manually.

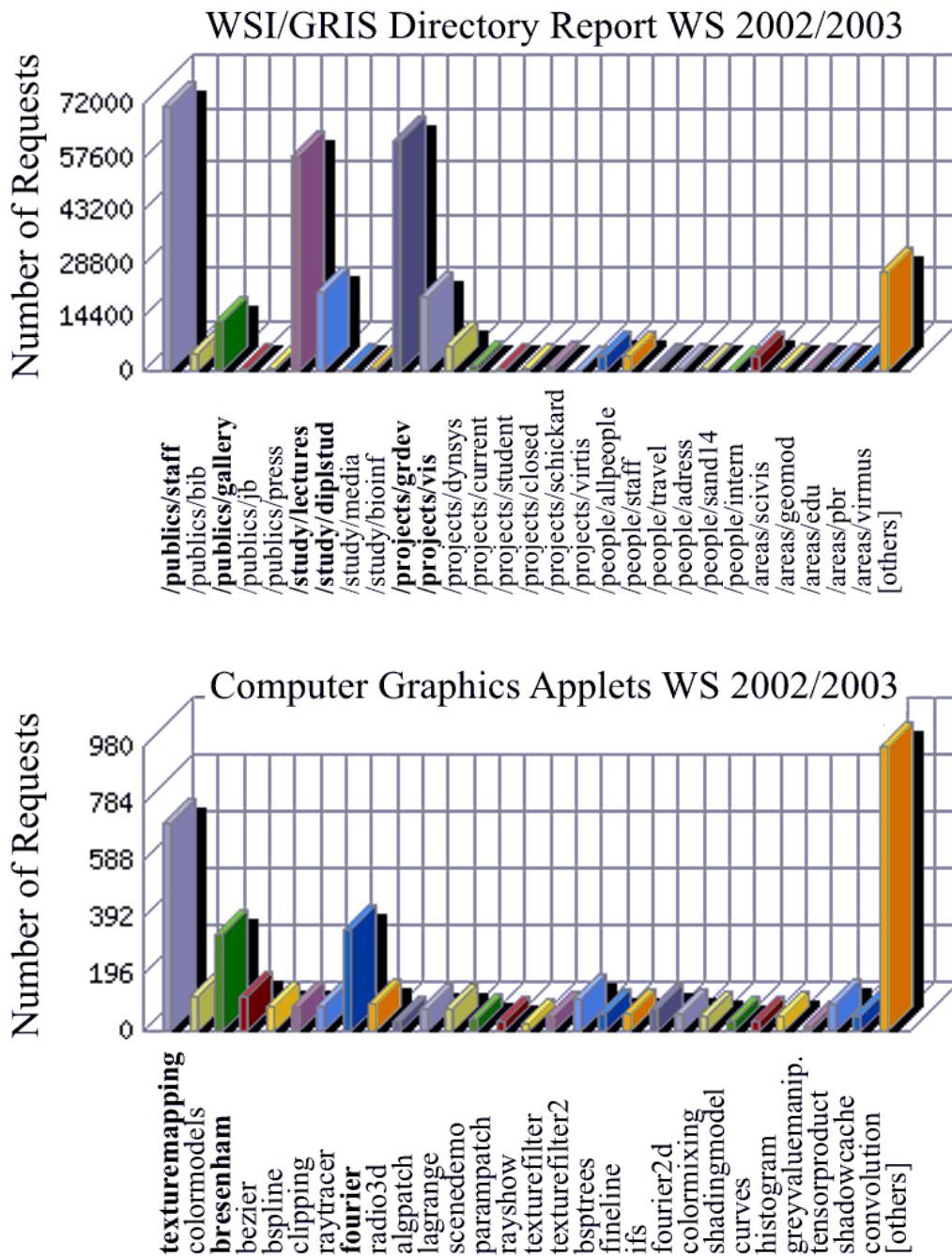


Figure 29: We evaluate Web server statistics to evolve courseware structuring. These diagrams display the use of our department's homepage (left side) and interactive learning objects of our Computer Graphics courseware (right side) in the winter semester 2002/2003. Visitors mostly requested staff member publications, lectures, and our courseware on Computer Graphics. Our most popular interactive learning objects visualize texture mapping, the Bresenham algorithm, and Fourier analysis. (Statistics include only Web pages and Java applet main classes. Robots, API documentation, and auxiliary pages were excluded. Keep in mind that we usually distribute our courseware on CD-ROM.)



## 5.2 Image Processing and Video Communications

### 5.2.1 Project

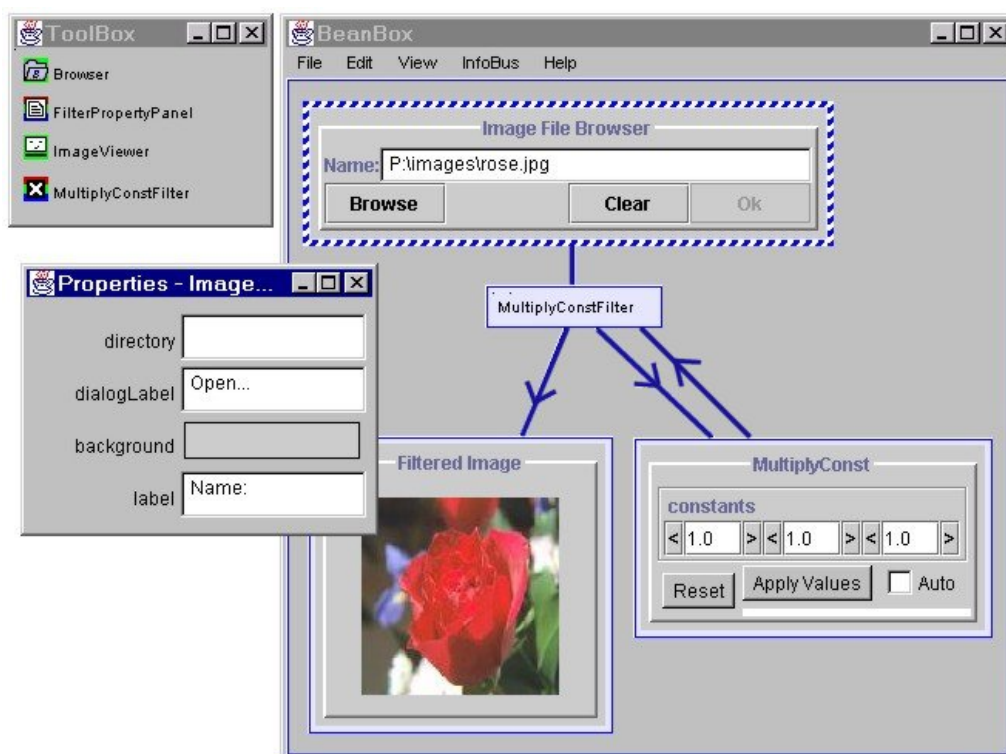
This second case study is meant to demonstrate how we apply component-based programming (2.4.2) in education (2.3.1). We present two complementary applications of interactive learning objects accompanying lectures at our institute. While the first one, a small courseware on *Image Processing* [Hanisch99] developed and employed in 1999 and 2000, focuses on visual programming of image filter chains, the second one, a courseware on *Image Communications* [Hanisch03b] held in 2001 and 2002, employs component programming to teach video processing. Each course took two hours per week and was taught by *Andreas Schilling*. Exercises required the same amount of time. Programming tasks in the first course represented a structure-based, top-down approach comprising, e.g. gap filling to complete filter loops. In contrast, exercises of the second course required low-level, bottom-up programming of entire components like the creation of a video renderer, filter, or stream. We compare these two approaches in the following sections.

### 5.2.2 Visual Programming

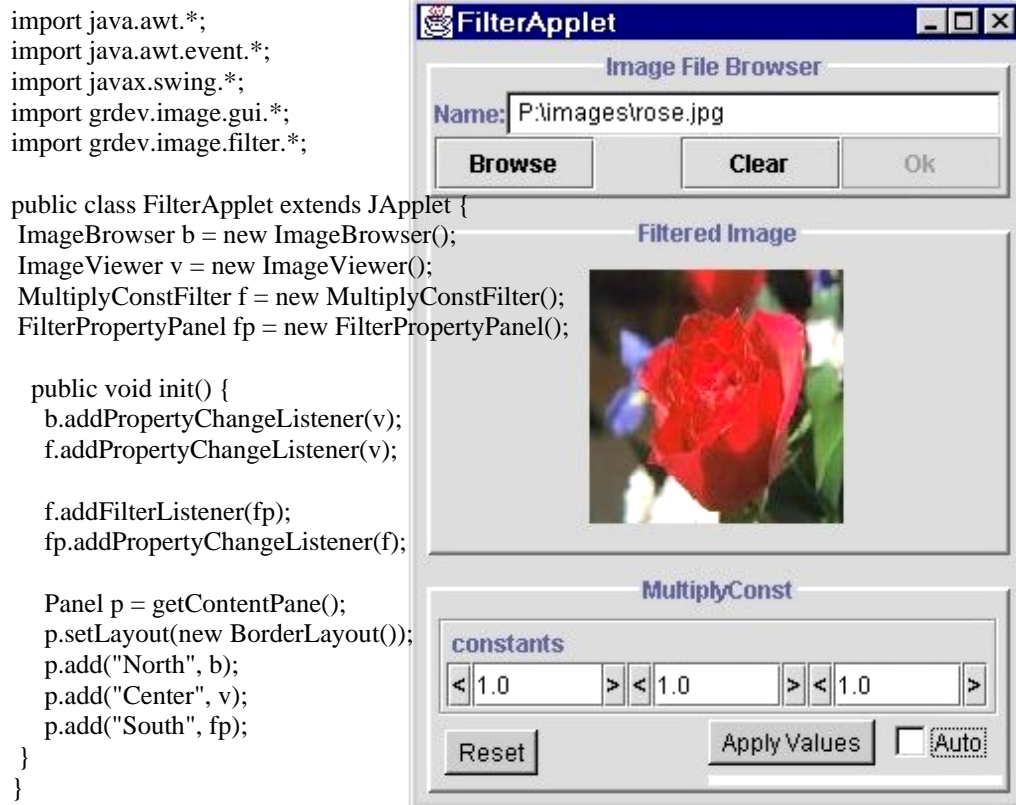
Our courseware on Image Processing includes interactive learning objects for teaching basics of image histograms and modifications, image operations and blending, discrete Fourier transformation, convolution kernels, image correction and reconstruction, decoding and encoding of image formats, edge detection, and image warping. Learning object development was preceded by the creation of a JavaBeans **toolkit** containing more than 50 image filters. Two-thirds of them could be based on native JAI (2.2.3, 3.2.1) filters; however, we reimplemented some of them for educational matters, i.e. to use them for gap-filling exercises on source code level. In addition, 15 GUI components offer image browsing, loading, editing, transcoding, storing, and interaction with histograms, filter kernels, transformations, and other properties. Learning objects are programmed visually by constructing a component data flow graphically in a builder tool (we used Sun's BeanBuilder, cp. the ESCOT project, 2.4.1). Therefore, each component sends property changes (2.4.2). Note that data flow naturally represents all stages from image retrieval to image processing and image viewing and storing as it matches the conceptual model of image filter chains.

Image filters provide an abstract property query mechanism, which we utilize to adapt the property panel to a specific filter. If the learner connects the property panel to a filter component, the panel queries all filter properties and sets up a matching GUI dynamically. That way, the learner may reuse a single property panel to interact with different filter components. For each property type (scalar, vector, kernel, image, etc.), we have registered a corresponding GUI editor, which will become instantiated, interlinked with the property, and embedded into the panel GUI whenever needed.

Each exercise assigns a **structure-based** task to the learner: given a set of input images with possible defects, how can we set up a filter chain producing the desired output? We introduced the toolkit incrementally. The first exercise offered a four-component subset, a browser, an RGB multiply filter, a viewer, and a property panel (see Figure 30). The learner simply had to connect them and interact with the RGB multipliers. The individual setups could be exported to a self-complete learning object, that is, a composite JavaBean. Figure 31 depicts the learning object resulting from the first exercise, together with the corresponding programming instructions. Subsequent exercises extended the set of components and asked the learner to design a more complex data flow. The fifth exercise for example asked the learner to develop a low pass image filter (see Figure 32). We attached appropriate filters and viewers for the discrete Fourier transformation and its inverse, diverse arithmetic image functions, 2D image functions and generic images, and support for real and imaginary values of complex images. In theory, low pass filtering entails a convolution with some image function, which maps on a multiplication in the frequency domain. The learner had to set up a data flow to normalize the input image, transform it to the frequency domain, visualize the log-magnitude of the complex result, multiply the transformed image with a generic image to cut unwanted frequencies, and transform it back to the time domain. Afterwards, he had to include and interpret different convolution filters, e.g. a Gaussian, box, circle, hole, or line function.



**Figure 30:** Visual programming in a builder tool enables the user to connect software components graphically by constructing a data flow. Here, the learner creates an interactive learning object that modifies an image's color channels.



**Figure 31: Programming interactive learning objects involves connecting software components by data flow and setting up the layout. Source code typically mirrors such clarity in structure. Only a modest number of programming instructions (left side) is needed to create the same learning object (right side) as created graphically in Figure 30. Visual programming, however, requires no technical knowledge regarding the specific programming language and environment.**

Visual programming allows the learner to understand the structure of a topic (here, algorithms) without having to become acquainted with the software environment. We generally started with graphical constructions to clarify the setup of the learning object and component dependencies; then, we went further into essential aspects of the topic by switching to **gap-filling** tasks in the components' source code. That way, we avoided confronting the learner with pure low-level programming, which would be rather time-consuming when dealing with specific image formats or filter performance.

Gap-filling programming tasks included mostly the development of appropriate filter loops iterating over all image pixel values. The second exercise for example requested a loop body implementing a threshold filter. Learners could base their solution on exemplary source code of an invert filter. Exercises such as local or global histogram equalization (rank filter and lookup filter, respectively) extended this basic approach. The final project addressed the reconstruction of a blurred image distorted by noise. Learners had to implement an iterative inverse filter expanding the Taylor series of an inverse Gaussian filter, and explore the impact of varying

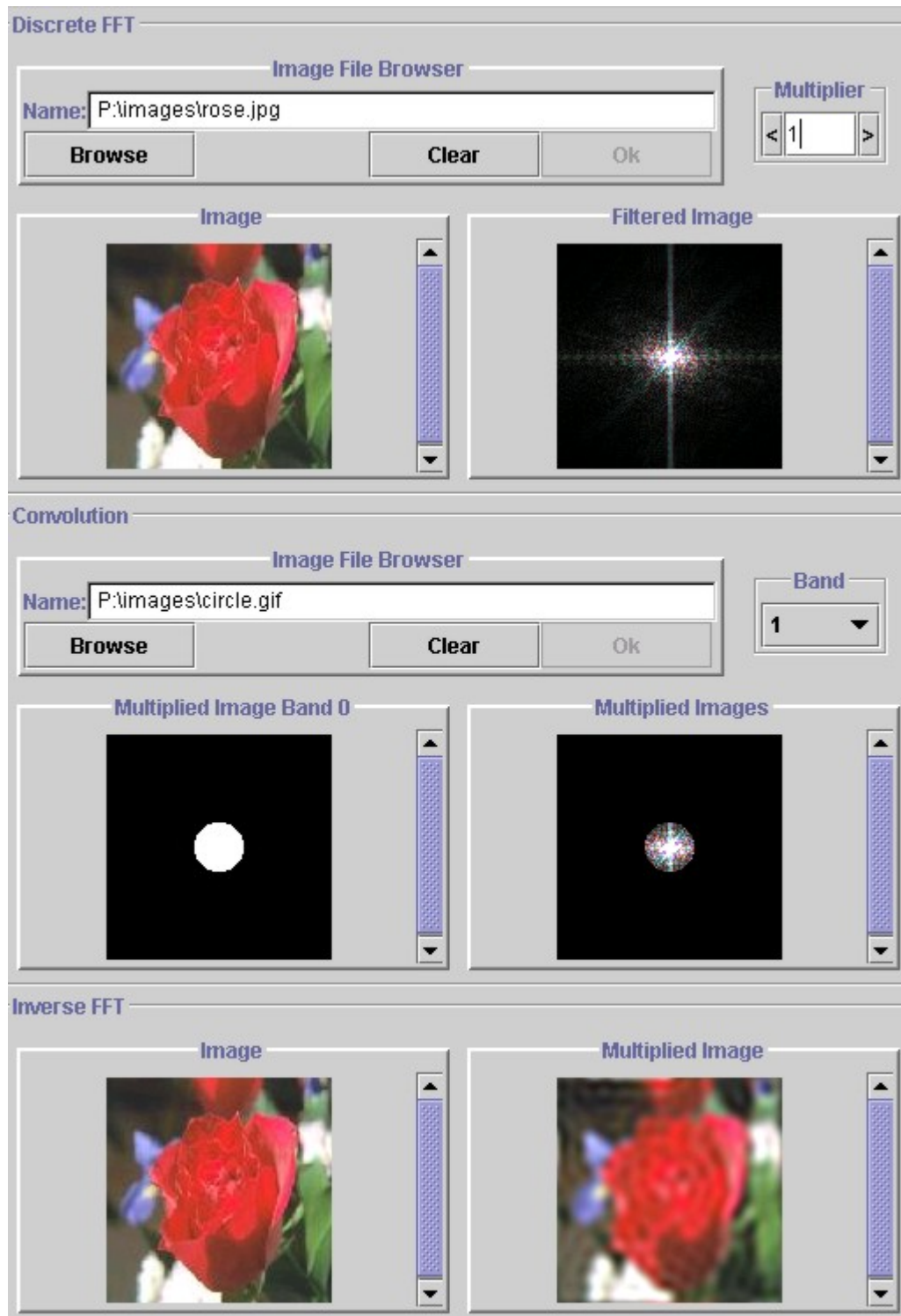
noise (the process diverges beyond some noise threshold). In this case, visual programming rather complicated the task, whereas component programming turned out to be the better alternative.

### 5.2.3 Component Programming

Our courseware on Video Communications shares many characteristics with our previously discussed courseware on Image Processing; in particular, we could reuse all image filters and GUI components for video filters working on single video frames. The great difference between these courses, however, lies in the importance of structure. In our Video Communications course, all applications share the same, typical data flow from the data source via the processor to the data sink. The processor decodes, filters, encodes, and renders a video stream. We do not focus on video filter chains – essentially, they do not differ from image filters – but individual components, i.e. data sources and sinks, codecs (decoder and encoder pairs), filters, renderers, de-/multiplexers, and de-/packetizers. Therefore, exercises naturally deal with component programming, that is, with the design and implementation of **new components** rather than with the composition and completion of existing ones.

We developed interactive learning objects for teaching the basics of time-based data sources (Web camera, video clips, image lists, screen, generic video), renderers for common formats (PAL/NTSC, JPEG compression, RGB/YUV color spaces, aspect ratios), frame- and time-based filters (transitions, image filters, and effects such as blue screen), and video streaming (RTP). Again, we spent most effort on creating a component **toolkit**. Our core framework for handling time-based media uses JMF (2.2.3, 3.2.1). Programming exercises were accompanied by the source code of a representative set of custom components. Each exercise dealt with a specific component type. We started by developing an image list protocol; the learner had to create a matching data source, encode it into a JPEG stream, and include frame-seeking functionality. We provided only an empty data source; the learner had to design his own test application for playing the video. A later exercise dealt with data formats and color spaces. Starting with a functional RGB renderer, we asked the learner to implement a new YUV renderer component (see Figure 21, p. 80) and to include it in the test program. A similar renderer handled widescreen.

Component programming requires fundamental knowledge of object-oriented, **low-level programming**. In contrast to visual programming, it enables the learner to understand all aspects of the learning content, including implementation details, special cases, and strategies for optimization. The overall structure of a problem stays in the background. For example, let us consider a more sophisticated stop-motion project. Learners created a storyboard, captured single frames or grabbed them from a video, reused the image list data source, and applied a blue-screen effect filter they had developed themselves. Some learners designed their custom filters. Next, we required the credits to be generated on the fly by some custom data source component. Learners finally concatenated their created clips and applied a custom transition. Programming took place on source code level, i.e. learners practiced Video Communications bottom-up.



**Figure 32:** This application of the discrete Fourier transformation implements a low-pass filter. Learners may visualize an image in time and frequency domain (first row), apply a convolution filter (second row), and compare the result with the original image (third row). We ask the learner to design the data flow and supply a matching convolution filter.

## 5.3 Scientific Visualization

### 5.3.1 Project

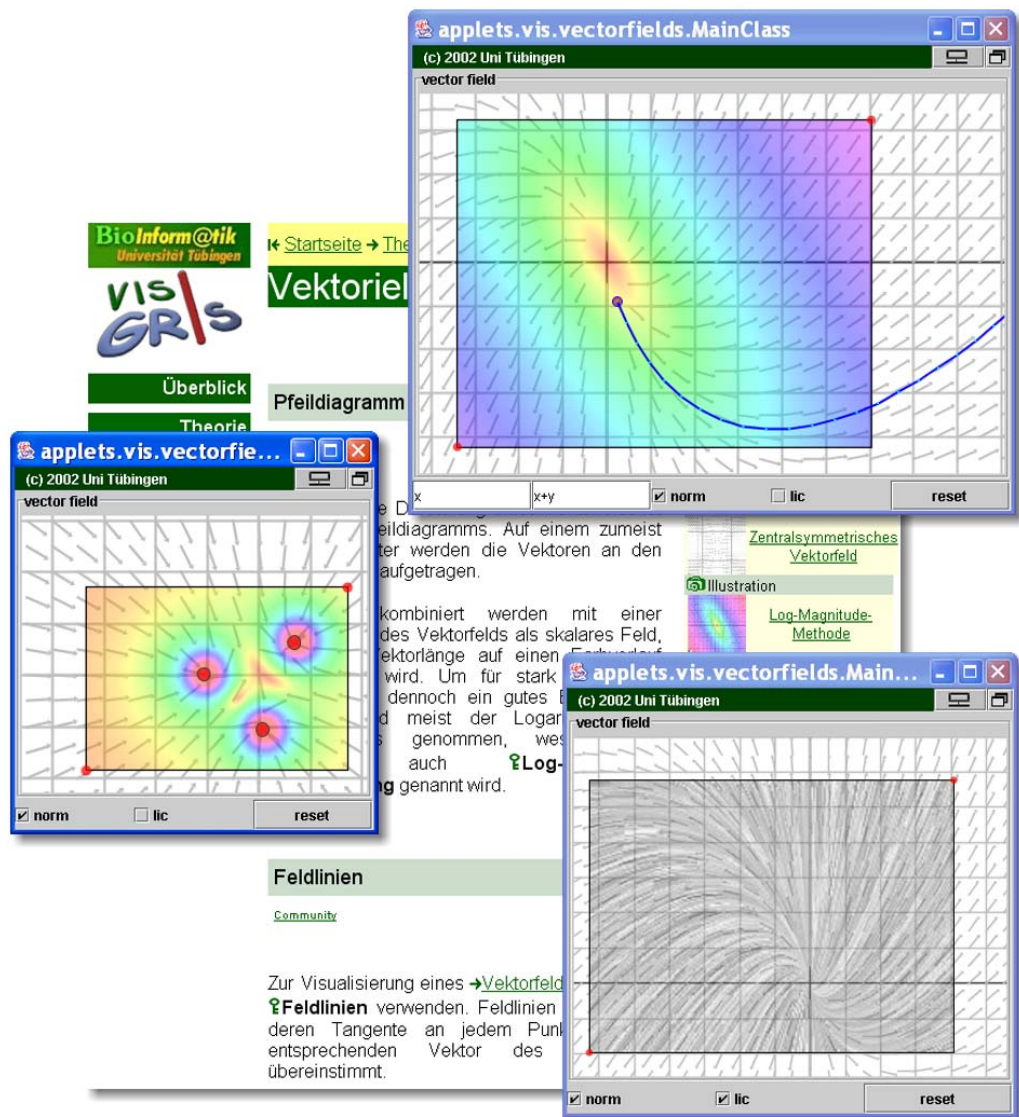
Our last case study demonstrates the use of scripting (3.3) and online wizards (4.2) in an interactive Web-based courseware on *Scientific Visualization* [Hanisch02b, Hanisch03a]. This two hours per week course (with exercises taking the same amount of time) was taught by *Stefan Gumhold* in 2002 and 2003. Scripting on the component- and within-component layer enabled us to create a fine-grained courseware interlinking. Learners can not only synchronize interactive learning objects with other courseware objects, but also adapt, rearrange, and exchange layout, design, and functionality (i.e. internal components and structure). Furthermore, we provided appropriate community support. Online wizards let learners personalize (annotate, rate, discuss, etc.) all courseware objects, including interactive ones, and work with the courseware collaboratively (reference, modify, share, etc.).

### 5.3.2 Scripting

We increased the effectiveness of our interactive Web-based courseware by realizing an adequate courseware interlinking. Before we depict a sample walkthrough in the domain of color vision (5.3.3), let us demonstrate how we overcame some typical problems an interactive learning object has to face in a Web-based environment, such as its isolated status, improper applied interactivity, and an overloaded GUI.

Figure 33 illustrates how we taught vector field visualization. At first, we included a course text providing the definition and basic properties of vector fields, and several illustrations of symmetric, radial, and potential fields (back window). We integrated a corresponding interactive learning object to let the learner experience these facts in a constructive manner. However, with black box interlinking (2.4.3) readings and interactivity would have stayed essentially separated. To realize a **fine-grained interlinking**, we applied our scripting approach and interlinked the course text with the interactive learning object's interior on the component and sub-component layer. For each statement and illustration of the course text, we embedded a script that adapts the object dynamically. The scripts insert particles into our scene graph to illustrate a potential field, redefine the vector field to be symmetric or radial, rearrange the GUI to include text fields for a function parser, or visualize a streamline (the surrounding windows show three different states). Other scripts exchange renderers that realize popular visualization techniques such as arrow plot, colorization, streamlines, and line integral convolution (LIC). As a result, learners may now switch back and forth between readings and interactivity freely, and adapt their interactive learning object to the statements and illustrations of the course text. Note that our scripts operate on the current object state, that is, they respect previously performed modifications.





**Figure 33:** Our courseware reuses this interactive learning object several times to illustrate different visualization techniques for vector fields. Programming exercises ask learners to include log-magnitude colorization, FastLIC, and their own algorithm.

Another major topic in Scientific Visualization is color vision. We have developed teaching material similar to that used for vector field visualization, and evaluated it with different target audiences. Besides our own course, we have supported a five-day IBM workshop targeting professional training, and an one-week student project for *Thomas Ertl's* lecture on Graphical-Interactive Systems at the University of Stuttgart. Exercises were carried out on site. We found that learners typically have difficulties in understanding basics of color perception [Beall96, Foley95], in particular the CIE color spaces and color defects (see Figure 34). CIE color spaces are based on the fact that the human eye has three types of color sensitive cones. It appears to be difficult for the learners to imagine the 3D horseshoe shape of all visible colors and describe related properties

(see Figure 34, first row). Note that any color can be expressed in terms of the two color coordinates (and a luminance term); matters of color (e.g. hue, saturation, and dominant wavelength), interpolation, and color gamuts are usually discussed only with 2D projections (ibid. second row, left side). Therefore, learners rarely have to reflect the 3D shape and related properties.

In our project, we asked the learners to simulate the effect of red, green, and blue color blindness (protanopes, deuteranopes, tritanopes) by calculating point projections onto a line (ibid. second row, right side). They had to verify their implementation with a given color test plate (ibid. third row). To solve this task, learners had to acquire an understanding of how to convert values between the CIE color spaces, how to set up a conversion matrix for a given color gamut within the CIE XYZ color space (using the phosphor values and white point of their monitor), and how to deal with out-of-gamut colors. Their solutions and questionnaires revealed that many learners made wrong assumptions if we provided them with course texts and non-interactive 2D illustrations only.

Based on such evaluations, we improved our interactive learning object in several aspects. A first enhancement with respect to the data flow was to allow graphical modification of all essential parameters, and to interconnect all related components – that is, to provide proper, bi-directional **interaction**. Learners can now compare a color's location in different color spaces, modify projection point and line, and apply arbitrary color test plates. We further included a common work flow in the learning process: visualize not only (1) one, fully saturated 2D projection of the 3D horseshoe shape, but also (2) various luminance layers, then (3) locate specific color values in the CIE color space, and (4) analyze corresponding RGB color values.

Our improvements led to an overloaded GUI; even before that, some learners had stated they could not identify properties described in theory within our learning object. Therefore, we outsourced functionality by embedding scripting instructions and hyperlinks into other courseware objects. Statements from the course text are now illustrated directly in the simulation; if learners read, for example, 'projection lines of tritanopes seem to be parallel, but intersect far away', a matching script scales down and translates the scene accordingly.

Finally, we modularized the simulation's content and now **introduce functionality gradually**. We used the fact that Java supports dynamic loading of classes and separated all Java 3D functionality from Java 2 and JAI functionality. Learners can now start working with 2D projections and color plates only (see Figure 34, second and third row). As they switch to theory and read about the 3D horseshoe shape, they may include the 3D visualization (ibid. first row) into their current simulation. The script modifies the layout to include a Java 3D canvas, three 2D graphs, and a table containing all numeric values. It also rearranges the simulation's data flow: learners may select a color in any canvas, input field, or table, and we update all other representations automatically.



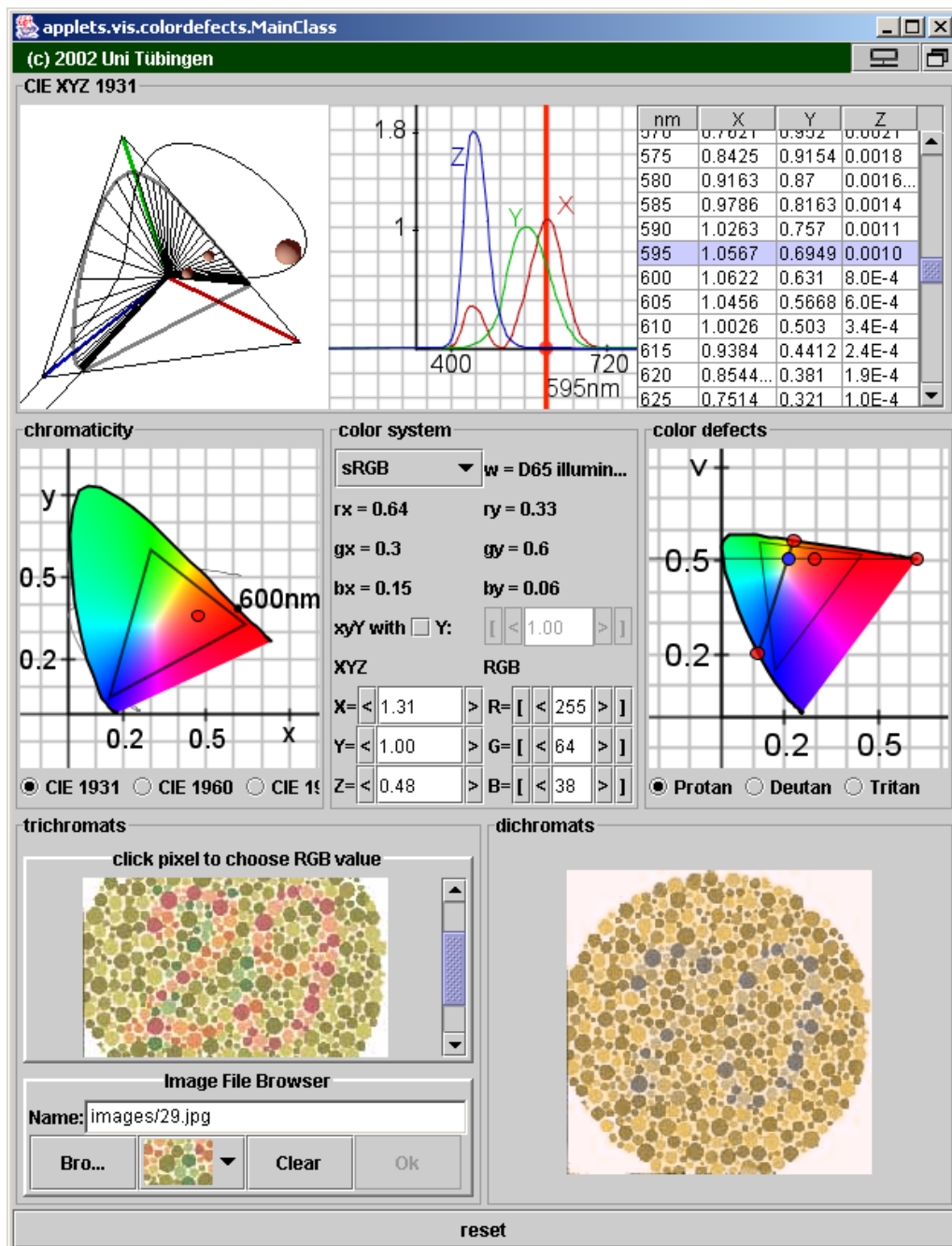


Figure 34: Teaching basics of color perception and color defects. The first row visualizes the 3D horseshoe shape of all visible colors. The second row illustrates various CIE chromaticity diagrams together with the effects of a color defect. The learner can simulate how protanopes, deuteranopes, and tritanopes see the world. The result is demonstrated in the third row.

### 5.3.3 Community

We close our case studies with a sample walkthrough a learner might take. In particular, we show how to perform self-tests, communicate with community members, and work with the courseware material.

Let us assume we seek information about color vision, for self-studies, lecture preparation, or revision. On the courseware homepage (<http://www.gris.uni-tuebingen.de/projects/vis>), we have the following options for navigating to the appropriate chapter in the course text: we can browse the chapter structure, search the index, browse the list of figures, or browse the list of interactive learning objects. We locate a chapter on color perception by searching the table of contents. By default, we obtain a novice version containing gap-filling text, exercises, and marginal notes. If we are tired of such optional information, we could switch to a condensed, expert version. The gap-filling text represents a **self-test**: some words of the course text are deleted and we are asked to choose a correct fill-in from a combo box. Three pictograms provide help; one formulates an explicit question whenever we move the mouse pointer over it, a second one starts up an online wizard providing one or more hints, and the last one leads us to an enquiry wizard to check our answers. We start reading the physical basics of light and color, and explore the described properties such as dominant wavelength, pureness, and luminance in the accompanying interactive learning object. While modifying these parameters, we simultaneously read the course text and activate some scripts (5.3.2) to locate specific color values in our simulation. We fill out gaps by, for example, naming a color that cannot be defined by a dominant wavelength (e.g. magenta) or describing a color by a linear, additive mixture of two spectral colors. After having similarly read and explored the basics of our visual system (e.g. the tristimulus theory), the CIE color spaces, and color defects, we conclude our self-test and enter the enquiry wizard. The wizard collects all questions belonging to the current chapter and evaluates our answers. In case of errors, the wizard reformulates the incorrectly answered questions, and provides more hints.

Next, we discuss our subject with other community members. In the marginalia, we find a discussion board (4.2.2) dedicated to the specific topic color perception. The boards are structured hierarchically, so if we are looking for a related topic, we can move to some board siblings (e.g. color models), or upwards to a more general board (e.g. visual cues). Other **communication** tools let us add short questions to the chapter, or answer the ones posed by other community members. We could further arrange a remote consultation with the tutor, in which we could synchronously discuss our problems with understanding the text, or participate in an online session with the interactive learning object. We prefer to express our impression of the content and start the rating wizard for the current course text. Feeling lazy, we skip the detailed review and just rate some qualities as (good or bad), and provide an overall grading. The author will receive our rating by email.

Finally, yet importantly, we start to **work with the content**, i.e. to personalize, modify, and extend it (4.2.2, 4.2.3). We add a personal note to the chapter, recording the insights we gained through communication and

the self-test. We make the note available for all community members, so that, if many of them rate our contribution as useful, it will become a static part of the courseware. One of the exercises included in the course text asks us to complete the background section with historical material about the visual system. Using an appropriate history wizard, we enter some of the contributions of *Isaac Newton* (1642-1726), who recognized that white light is made up of all the colors that we can see, and *Hermann von Helmholtz* (1821-1894), originator of the variables hue, saturation, and brightness, and advocator of the three-color system. If we take part in one of the student projects, we are further entitled to modify or extend particular passages of the course text as part of the assignment. Note that during editing we may reference any other learning object or part of it (e.g. a learning object, or a script). Other exercises provide us with the core CIE XYZ data set and ask us to visualize it, or to set up a script to display the color gamut of our own monitor in the visualization. Programming exercises require us to study the simulation's source code, and enhance the algorithm handling out-of-gamut values, or implement the transition from trichromatism to dichromatism. Finally, we are assigned the task of designing our own online test for color perception defects, which will become integrated into the courseware.



## 6 Conclusion & Directions for Future Work

We have traveled the path from the first teaching machine to learning management systems, and interrelated paths of hypermedia and graphical user interfaces that led to the Web and the Desktop. We have met visions, and grand challenges. Ours is that of interactive Web-based courseware, which we witness as an integral part of our teaching and learning, and which we would like to bring to the masses. We had to state that the basic concept of interactivity, hypermedia, and courseware – the idea of an highly interactive medium with means for collaboration, personalization, and working with the content – is obscured due to absent technology and concrete form.

Learning technologies have specified learning objects and learning management systems. However, few methodologies exist for interactive learning objects, and not a single one for highly interactive ones. While technology provided direct manipulation, software components, and scripting, didactics and cognitive theories established adequate mental models, discovery learning, and microworlds. We combined these approaches into an MVC Interactivity and formulated symbolical interactivity levels in software terminology. As we required direct manipulation not only for object view and parameters, but also for functionality (components and structure), we presented visual Drag & Drop Scripting. Our ORC-SG design pattern provided a more granular MVC component model with respect to construction and interactive, graphical content.

Through examples, we discussed an application of current learning technology standards and illustrated some of our most urgent needs. To include issues of interactivity and interoperability into a digital library, we presented the concept and prototype of a scripting database, together with online wizards enabling community members to modify interactive learning objects online and interlink them with other learning objects in a graduated manner. Certainly, our approach marks just the beginning of collaborative authoring of interactive learning objects, an area that is not yet explored. Effective systems will have to include appropriate authoring tools. We are thinking of a Web-enabled integrated development environment (IDE) letting community members program components and scripts online, possibly visually. We also need Web-based component models standardizing an object's programming interface and interoperability.

Subjects like these must become included into learning object metadata and the runtime environment of a learning management system; then, we might be able to specify, retrieve and adapt interactive learning objects adequately, and evaluate user interactions on component and sub-component layer. *Andries van Dam* wants interactive objects to become clip models [vanDam02, CRA02], with the same characteristics as conventional clip art.

The desired flexibility naturally requires collaborative efforts of the educational community. Roschelle [Roschelle98] asked how to anticipate

and support an emerging community of practice around component software and customizable curricula. Our Web framework represents a prototype for such a social architecture. We manage learning objects, components and scripts in the courseware's repository, and let community members share, annotated, rate, and extend them. Generator templates and online wizards support them both in interlinking, and authoring. We demonstrated the benefits of such an integrated system with examples from our courseware. Yet, we had to postpone some ideas for future work. Firstly, we did not mention the application of our layered database model to software components. By providing alternative components, we could render interactive objects useful for adaptive systems [Brusilovsky96]; besides multilinguality, we could precisely adapt presentation, structure, and depth of information.

Secondly, concerning component reuse, we discussed only the tip of the iceberg. In this project, we reused design, geometry, and structure information to facilitate the development of interactive learning objects, and adapt them to other courseware objects. Drag & Drop Scripting allowed us to transport such information beyond the browser barrier. We would like software components to become intelligent in the sense that they should be evolved by community members in multiple microworlds or online games, and transferred to different setups with their current characteristics. Intelligent components must therefore comprise an extendable, context-sensitive set of attributes, actions, and functionality. Lastly, they will be shared and distributed in a digital library, and billed on component base.







## Abbreviations

Systems and projects are located in the Name Index (next page).

API	Application Programming Interface
CGI	Common Gateway Interface
CIE	Commission Internationale de L'Eclairage
CSCW	Computer Supported Cooperative Work
FAQ	Frequently Asked Questions
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
JAI	Java Advanced Imaging
JDBC	Java Database Connectivity
JMF	Java Media Framework
JPEG	Joint Photographic Experts Group
LIC	Line Integral Convolution
LMS	Learning Management System
LOM	Learning Object Metadata
MVC	Model View Controller
NTSC	National Television System Committee
ODBC	Open DataBase Connectivity
ORC-SG	Object, Renderer, Constraint, Scene Graph, and GUI
PAL	Phase Alternating Line
PDF	Portable Document Format
RGB	Red, Green, and Blue
RTP	Real Time Transport Protocol
SCORM	Sharable Content Object Reference Model
SMET	Science, Mathematics, Engineering, and Technology
SQL	Structured Query Language
TCP	Transfer Control Protocol
URL	Uniform Resource Locator
UNL	Uniform Network Location
WIMP	Windows, Icons, Menus, and Pointer device (mouse)
WYSIWYG	What You See Is What You Get
YUV	Luminance and Two Chrominance Signals

## Name Index

The following names are typed in italics in the main text.

### A

ADL 43  
AICC 43  
Amaya 30  
Andrews, Keith 24, 29, 39  
ARIADNE 43, 51  
ARPA 24

### B

BALSA 54  
Baumgartner, Peter 32, 37  
Berners-Lee, Tim 29  
BibTex 103  
Bitzer, Donald 33  
Borning, Alan 26, 53  
Boxer 27  
Bravo 26  
Bruner, Jerome 33  
Bush, Vannevar 24

### C

CGEMS 12  
Christian, Wolfgang 55  
Cinderella 23, 54  
Computing Research Association 11

### D

Dexter Hypertext Reference Model 28, 55  
diSessa, Andrea 27, 46  
DoD 24, 33  
Duval, Erik 42, 47  
Dynabook 13, 30

### E

ED-MEDIA 11  
Electronic Webmaster 100  
ELS 26  
Engelbart, Douglas 25, 28  
EOE 50  
Ertl, Thomas 111  
ESCOT 49, 56  
E-Slate 48, 56  
Exploratories 39, 46

### F

Flash 30  
FRESS 25, 35

### G

Gentle 40, 41  
Gumhold, Stefan 110

### H

Halasz, Frank 26, 28  
HES 25  
HTML 29, 30  
HTTP 29  
Hypercard 28, 35  
Hyper-G 29  
HyperTies 18  
Hyperwave 29, 38, 40

### I

IEC 42  
IEEE 42  
IMS 38, 43  
Ingalls, Daniel 26  
Intermedia 27, 35  
Internet Explorer 30

### J

Java 19, 26, 31  
Java 3D 31, 48, 54, 61, 67  
Java Advanced Imaging 31, 61, 105  
Java Media Framework 31, 61, 108

### K

Kay, Alan 11, 13, 19, 26, 34  
Klein, Reinhard 100

### L

LEGO Mindstorms 34  
Licklider, Joseph 24  
Logo 34  
LOM 22  
LTSC 42, 43

### M

MANKIND 29  
MathWorlds 56  
Maurer, Hermann 11, 29, 35, 39, 41  
Memex 24  
MERLOT 51  
Merrill, David 33  
Meyrowitz, Norman 27, 28, 37  
Mozilla 30

**N**

Nelson, Ted 25, 35  
Netscape Navigator 30  
Newton, Isaac 115  
NLS 25  
NoteCards 26  
NSF 45

**P**

Papert, Seymour 34  
Physlets 55  
Piaget, Jean 34  
PROMETHEUS 43  
Pygmalion 26

**Q**

Quicktime 30

**R**

RealPlayer 30  
Roschelle, Jeremy 28, 47, 56

**S**

Schulmeister, Rolf 22, 36, 60  
SCORM 36, 43, 78  
Servlets 90  
Shneiderman, Ben 15, 18, 19  
Shockwave 27, 30  
Simonyi, Charles 26  
Sims, Rod 19  
Sketchpad 18, 25  
Skinner, Burrhus Frederic 33  
Smalltalk 19, 26  
SMIL 30  
Smith, David 26  
SQL 38

Squeak 26  
Star 27  
Straßer, Wolfgang 100  
Sutherland, Ivan 18, 25  
SVG 30

**T**

Tesler, Larry 26  
ThingLab 26  
Thorndike, Edward 33  
TICCIT 33

**V**

van Dam, Andries 11, 25, 35, 46, 47, 117  
von Helmholtz, Hermann 115  
von Neumann, John 24

**W**

W3C 30  
Web/Comp 46  
WebCT 37, 41  
WebDAV 30  
World Wide Web 29

**X**

Xanadu 25  
Xerox PARC 26  
XML 30

**Y**

Yankelovich, Nicole 27

**Z**

Zuse, Konrad 24

## Subject Index

The following subjects are typed bold-faced in the main text.

### A

adapter 61, 66  
 adaptive system 15  
 added value 15  
 alternative points of view 74  
 annotation 94  
 authoring tools 38

### B

behaviorism 33  
 behaviors 67  
 black boxes 38  
 browser 29  
 building block 19, 42

### C

classroom 75  
 cognitivism 33  
 communication 114  
 community 40  
 compatibility 96  
 components 19, 26, 50, 52, 87, 108  
 constraint 26, 53, 62  
 construction 22  
 constructivism 34  
 container 27, 70  
 content 88  
 context 26  
 context help 94  
 courseware 13, 36

### D

data flow 62  
 database 38  
 decentralization 77  
 design 88  
 dictionaries 85  
 digital libraries 45  
 direct manipulation 19  
 discussion board 94  
 drag & drop scripting 79

### E

external representation 21

### F

functionality 75, 112

### G

generator 40, 76, 86

granularity 46, 70  
 graphical user interface 65  
 guidelines 21

### H

highly interactive learning object 13, 60, 95  
 Human-Computer Interaction 18  
 hyperlinks 24  
 hypermedia 13, 24, 25, 35, 37  
 hypermodel 56

### I

illustration 73  
 Image Communications 105  
 Image Processing 105  
 incrementally given data 85  
 integration of information 21  
 interaction 18, 20, 22, 25, 112  
 interactive constructs 19  
 interactivity 18  
 interlinking 87, 110  
 interoperability 27, 44

### L

layers 84, 100  
 learning management system 36  
 learning object 42  
 learning technology standards 22, 42, 78  
 learning theory 32  
 link consistency 84  
 link database 26  
 look & feel 65

### M

member status 93  
 messages 26, 53  
 metadata 22, 44, 51, 68, 77, 85  
 Model View Controller 53  
 modelessness 26  
 multimedia 18

### N

network 24, 75

### O

object 63  
 output-sensitive 64

**P**

plug-in 30, 65  
programming 19, 54, 107, 108  
properties 63

**R**

rating system 94  
renderer 63  
repositories 25, 46, 50  
review process 51

**S**

scene graph 54, 66, 67  
Scientific Visualization 110  
scripting 28, 48, 55, 72, 95  
search 103  
self-test 95, 114  
server 76  
structure 88, 106

synchronization 73

**T**

templates 39, 72, 86, 91, 101  
toolkit 105, 108

**U**

upload 103  
usability 22

**V**

visual programming 26

**W**

Web-based authoring 55, 91, 103, 114  
WIMP 27  
within-component layer 29  
wizards 41, 90, 95, 101



## Bibliography

References are in alphabetical order of first author. For any given Web resource (URL), the last date of access was November 20, 2003.

[ADL01a] Advanced Distributed Learning, *Sharable Content Object Reference Model (SCORM), Version 1.2, The SCORM Overview*, October 2001.

URL: [http://www.adlnet.org/screens/shares/dsp\\_displayfile.cfm?fileid=480](http://www.adlnet.org/screens/shares/dsp_displayfile.cfm?fileid=480)

[ADL01b] Advanced Distributed Learning, *Sharable Content Object Reference Model (SCORM), Version 1.2, The SCORM Content Aggregation Model*, October 2001.

URL: [http://www.adlnet.org/screens/shares/dsp\\_displayfile.cfm?fileid=476](http://www.adlnet.org/screens/shares/dsp_displayfile.cfm?fileid=476)

[ADL01c] Advanced Distributed Learning, *Sharable Content Object Reference Model (SCORM), Version 1.2, The SCORM Run-Time Environment*, October 2001.

URL: [http://www.adlnet.org/screens/shares/dsp\\_displayfile.cfm?fileid=483](http://www.adlnet.org/screens/shares/dsp_displayfile.cfm?fileid=483)

[Aldrich98] Aldrich, F. Rogers, Y., Scaife, M., *Getting to grips with 'interactivity': helping teachers evaluate the educational value of CD-ROMs*, British Journal of Educational Technology. 29(4):321-333, 1998.

[Ambron88] Ambron, S., and Hooper, K., *Interactive multimedia*, Redmond, Microsoft, 1988.

[Andrews94] Andrews, K., Kappe, F., Maurer, H., Schmaranz, K., *On Second Generation Hypermedia Systems*, Journal of Universal Computer Science 0(0):127-135, November 1994.

URL: [http://www.jucs.org/on\\_second\\_generation\\_hypermedia\\_systems](http://www.jucs.org/on_second_generation_hypermedia_systems)

[Andrews96] Andrews, K., *Browsing, Building, and Beholding Cyberspace, New Approaches to the Navigation, Construction, and Visualisation of Hypermedia on the Internet*, Ph.D. thesis, Graz University of Technology, 1996.

URL: <http://www2.iicm.edu/keith-phd>

[Apache00] Apache Software Foundation, *The Apache Jakarta Project: Struts User Guide*, 2000-2003.

URL: <http://jakarta.apache.org/struts/userGuide>

[ARIADNE02] ARIADNE Foundation, *ARIADNE Educational Metadata Recommendation - V3.2*, February 2002.

URL: <http://www.ariadne-eu.org>

[Azevedo01] Azevedo, F. S., *EOE Profile*, in: diSessa, A. (ed.), the Web/comp project, March 2001.

URL: <http://dewey.soe.berkeley.edu/~boxer/webcomp>

[Bacher97] Bacher, C., Müller, R., Ottmann, T., Will, M., *Authoring on the Fly: a new way of integrating telepresentation and courseware production*, Proc. of ICCE '97, Kuching (Malaysia), December 1997.

- [Barker02] Barker, p. , Rebelsky, S., *Proceedings of ED-MEDIA 2002. World Conference on Educational Multimedia, Hypermedia & Telecommunications*, AACE, June 2002.  
URL: <http://www.diffuse.org/ED-Media-02.html>  
Abstracts of Opening Keynotes
- [Baumgartner92] Baumgartner, p. , Payr, S., *Lernen mit Software*, 2nd ed., Innsbruck, StudienVerlag, 1992.
- [Baumgartner98] Baumgartner, p. , Payr S., *Learning with the Internet. A Typology of Applications*, Proceedings of ED-MEDIA 98 - World Conference on Educational Multimedia and Hypermedia, Charlottesville, AACE, pp.124-129, 1998.
- [Baumgartner02] Baumgartner, p. , Häfele, K., Häfele, H., *E-Learning: Didaktische und technische Grundlagen*, Sonderheft des bm:bwk, CDAustria, Mai 2002.  
URL: <http://www.peter.baumgartner.name>
- [Beall96] Beall, J. E., Doppelt, A. M., Hughes, J. F., *Developing an Interactive Illustration: Using Java and the Web to Make It Worthwhile*, Computer Graphics (Proceedings of 3D and Multimedia on the Internet, WWW and Networks), Bradford, UK, 1996.
- [Bell95] Bell, G., Parisi, A., Pesce, M., *The Virtual Reality Modeling Language, Version 1.0 Specification*, November 1995.  
URL: <http://www.web3d.org/VRML1.0>
- [Bentley97] Bentley, R., Appelt, W., Busbach, U., Hinrichs, E., Kerr, D., Sikkell, K., Trevor, J., Woetzel, G., *Basic Support for Cooperative Work on the World Wide Web*, International Journal of Human Computer Studies 46:827-846, 1997.
- [BernersLee89] Berners-Lee, T., *Information Management: A Proposal*, CERN, Geneva, 1989.  
URL: [www.w3.org/History/1989/proposal.html](http://www.w3.org/History/1989/proposal.html)
- [BernersLee99] Berners-Lee, T., *Weaving the Web*, HarperCollins, 1999.
- [Birbilis00] Birbilis, G., Koutlis, K., et al., *E-Slate: A software architectural style for end-user programming*, presented at the 22<sup>nd</sup> International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, 2000.  
URL: <http://e-slate.cti.gr>  
E-Slate
- [Bitzer70] Bitzer, D.L., Skaperdas, D., *The Economics of a Large Scale Computer Based Educational System: Plato IV*, in: Holtzman, W., Computer Assisted Instruction, Testing and Guidance, pp.17-29, New York, Harper and Row, 1970.
- [Borning81] Borning, A., *The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory*, ACM Transactions on Programming Languages and Systems 3(4):353-387, October 1981.



- [Bothun97] Bothun, G., Kevan, S., Micklavzina, S., Mason, D., *Networked Physics Curriculum: From Static Web to Dynamic Java*, International Journal of Modern Physics 8:79-91, 1997.
- [Blumstengel99] Blumstengel, A., *Entwicklung hypermedialer Lernsysteme*, Ph.D. thesis, University of Paderborn, Wissenschaftlicher Verlag Berlin, 1999.  
URL: <http://dsor.uni-paderborn.de/de/forschung/publikationen/blumstengel-diss>
- [Borsook91] Borsook, T. K., Higgenbotham-Wheat, N., *Interactivity: What is it and what can it do for computer-based instruction*, Educational Technology Research and Development, pp.11-17, October 1991.
- [Bush45] Bush, V., *As We May Think*, The Atlantic Monthly, 176(1):101-108, July 1945.  
URL: <http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm>
- [Brown84] Brown, M. H., Sedgewick, R., *A system for algorithm animation*, Computer Graphics, 18(3):177-186, 1984.
- [Bruner61] Bruner, J. S., *The act of discovery*, Harvard Educational Review, 31(1):21-32, 1961.
- [Bruner66] Bruner, J. S., *Towards a theory of instruction*, Cambridge, Harvard University Press, 1966.
- [Brusilovsky96] Brusilovsky, p. , *Methods and techniques of adaptive hypermedia*, User Modeling and User-Adapted Interaction, 6(2-3):87-129, 1996.
- [Brusilovsky98] Brusilovsky, p. , *Adaptive Educational Systems on the World Wide Web: A Review of Available Technologies*, Proc. of workshop WWW-Based Tutoring at 4<sup>th</sup> International Conference on Intelligent Tutoring Systems (ITS'98), 1998.
- [Carr99] Carr, L., Hall, W., De Roure, D., *The Evolution of Hypertext Link Services*, ACM Computing Surveys 31(4), December 1999.
- [Conklin87] Conklin, J., *Hypertext: An introduction and Survey*, IEEE Computer, 20(9):17-41, 1987.
- [Christian00] Christian, W., Belloni, M., *Physlets*, Prentice Hall, NJ, 2000.  
URL: <http://webphysics.davidson.edu/Applets/Applets.html>
- [Conway97] Conway, M. J., *Alice: Easy-to-Learn 3D Scripting for Novices*, Ph.D. thesis, University of Virginia, 1997.  
URL: <http://www.alice.org>
- [CRA02] Computing Research Association, *Grand Challenge 3: Provide a Teacher for Every Learner*, in: Grand Research Challenges in Information Systems, Final Report of the CRA Conference on "Grand Research Challenges" in Computer Science and Engineering, Warrenton, Virginia, pp.17-22, CRA, 2003.  
URL: <http://www.cra.org/Activities/grand.challenges>

- [Cunningham01] Cunningham S., Bailey M. J., *Lessons from Scene Graphs: Using Scene Graphs to Teach Hierarchical Modeling*, Computers & Graphics, 25(4):703-711, 2001.
- [Dietinger98] Dietinger, T. Maurer, H., *Gentle – (General Networked Training and Learning Environment)*, Proc. ED-MEDIA98, Juni 1998.
- [diSessa85] diSessa, A. A., *A Principled Design for an Integrated Computational Environment*, Human-computer interaction, 1:1-47, 1985.
- [diSessa86a] diSessa, A. A., *Artificial Worlds and Real Experience*, Instructional Science 14:207-227, 1986.
- [diSessa86b] diSessa, A. A., Abelson, H., *Boxer: A Reconstructible Computational Medium*, Communications of the ACM, 29(9):859-868, 1986.
- [diSessa97] diSessa, A. A., *Open toolsets: New ends and new means in learning mathematics and science with computers*, in Pehkonen, E. (ed.), Proc. of the 21st Conference of the International Group for the Psychology of Mathematics Education 1:47-62. Lahti, Finland, 1997.
- [diSessa01] diSessa, A. A. (ed.), *the Web/comp project*, January 2001.  
URL: <http://www.soe.berkeley.edu/~boxer/webcomp/>
- [Duval95] Duval, E., Olivié, H., O'Hanlon, p. , Jameson, D. G., *HOME: an Environment for Hypermedia Objects*, Journal of Universal Computer Science 1(5):269-291, Mai 1995.  
URL: [http://www.jucs.org/jucs\\_1\\_5/home\\_an\\_environment\\_for](http://www.jucs.org/jucs_1_5/home_an_environment_for)
- [Duval01] Duval, E., Forte, E., Cardinaels, K., et al., *The ARIADNE Knowledge Pool System: a Distributed Digital Library for Education*, Communications of the ACM, 44(5):72-78, ACM, New York, Mai 2001.  
URL: <http://www.ariadne-eu.org>
- [Duval02] Duval, E., Ternier, S., *Learning Object Metadata: A Hands on Workshop*, ED MEDIA 2002: World Conference on Educational Multimedia, Hypermedia and Telecommunications, Denver, Colorado, AACE, June 2002.  
URL: <http://www.diffuse.org/ED-Media-02.html>
- [Dix96] Dix, A., *Challenges and Perspectives for Cooperative Work on the Web*, Proceedings of the ERCIM workshop on CSCW and the Web, Sankt Augustin, February 1996.
- [Eberhardt99] Eberhardt, B., Gürçay, H., Hanisch, F., Hüttner, T., Licht, O., Nill, B., *Books and Devices from the Old – their Renaissance in Computer Graphics*, Eurographics'99, Short paper, 1999.
- [Engelbart68] Engelbart, D. C., *The original 90-minute live public demonstration of the online system NLS*, in: Engelbart Collection in Special Collections of Stanford University, December 1968.  
URL: <http://sloan.stanford.edu/mousesite/1968Demo.html>

- [Figueiredo03] Figueiredo, F. C. , Eber, D. E., Jorge, J. A., *A Refereed Server for Educational CG Content*, The annual conference of the European Association for Computer Graphics (Eurographics 2003), University of Granada, Granada, September 2003.
- [Foley95] Foley J., van Dam, A., Feiner S. K. , Hughes J. F., *Computer graphics, principles and practice*, 2nd ed., Addison-Wesley, July 1995.
- [Gamma94] Gamma, E., et al. *Design patterns elements of reusable object-oriented software*, Reading, MA, Addison-Wesley, 1994.
- [GAO03] United States General Accounting Office, *MILITARY TRANSFORMATION: Progress and Challenges for DOD's Advanced Distributed Learning Programs*, Report to Congressional Committees, Washington DC, February 2003.  
URL: [www.gao.gov/cgi-bin/getrpt?GAO-03-393](http://www.gao.gov/cgi-bin/getrpt?GAO-03-393)
- [Halasz88] Halasz, F. G., *Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems*, Communications of the ACM 31(7):836-852, 1988.
- [Halasz90] Halasz, F. G., Schwartz, M. D., *The Dexter Hypertext Reference Model*, Proc. of the Hypertext Standardization Workshop by National Institute of Science and Technology (NIST), January 1990. Reprinted in: Communications of ACM, 37(2):30-39, February 1994.
- [Hanisch99a] Hanisch, F., Klein, R., Straßer, W., *Ein Web-basierter Computergraphik-Kurs im Baukastensystem*, in: Engeli, M., Homann, J. (eds.), *Virtuelle Organisation und Neue Medien - Workshop GeNeMe99*, pp.255-270, Eul-Verlag, Lohmar, 1999.
- [Hanisch00a] Hanisch, F., Klein, R., *Challenges in the Development of Web-based Courseware using Virtual Experiments*, in: Hoffmann, M. H. W. (ed.), *Innovations in Education for Electrical and Information Engineering (EIE) – Proc. of the 11<sup>th</sup> annual conference of the EAEEIE*, pp.119-124, Universität Ulm, 2000.
- [Hanisch00b] Hanisch, F., *Basic Requirements for Interactive Web-based Courseware*, in: Auer, M. (ed.), *Interactive Computer aided Learning Applications and Experiences*, Carinthia Tech Institute, School of Electronics, Villach, 2000.
- [Hanisch01a] Hanisch, F., *Scripting and Cooperation for Interactive Web-based Courseware*, in: 3<sup>rd</sup> International Conference of New Learning Technologies (NLT), University of Applied Sciences of Western Switzerland, Fribourg, September 2001.
- [Hanisch01b] Hanisch, F., Eberhardt, B., Nill, B., *Reconstruction and virtual model of the Schickard calculator*, Journal of Cultural Heritage, pp.335-340, Éditions scientifiques et médicales Elsevier SAS, 2001.
- [Hanisch02a] Hanisch, F., *Authoring and Linking of Highly Interactive Content within Web-based Courseware*, Networked Learning in a

Global Environment: Challenges and Solutions for Virtual Education (NL 2002), Technical University of Berlin, 2002.

[Hanisch02b], Hanisch, F., *Using Scripting to Increase the Impact of Highly Interactive Learning Objects*, ED-MEDIA 2002: World Conference on Educational Multimedia, Hypermedia & Telecommunications, Denver, Colorado, AACE, 2002.

[Hanisch03a] Hanisch, F., Straßer, W., *Adaptability and Interoperability in the Field of Highly Interactive Web-Based Courseware*, Computer & Graphics 27(4):647-655, 2003.

URL: <http://www.gris.uni-tuebingen.de/projects/vis>  
Scientific Visualization courseware

[Hanisch03b] Hanisch, F., Straßer, W., *Drag & Drop Scripting: How To Do Hypermedia Right*, in: Cunningham, S., Martin, D. (eds.), Eurographics Education Presentations (EG'03), Eurographics Association, 2003.

[Helic00] Helic, D., Maurer, H., Scherbakov, N., *Web Based Training: What do we expect from the system*, Proc. of ICCE 2000, pp.1689-1694, Taiwan, 2000.

URL: [http://www.iicm.edu/iicm\\_papers/web\\_based\\_training.doc](http://www.iicm.edu/iicm_papers/web_based_training.doc)

[Hewett92] Hewett, T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., Verplank, W., *ACM SIGCHI Curricula for Human-Computer Interaction*, Report of the ACM SIGCHI Curriculum Development Group, ACM, 1992.

URL: <http://www.acm.org/sigchi>

[HIM99] Hyperwave Information Management, Inc., *Hyperwave Administrator's Guide, Hyperwave Information Server, Version 5.1*, Westford, USA, September 1999.

URL: <http://www.hyperwave.com>

[HIM00] Hyperwave Information Management, Inc, *Hyperwave: eLS Expert's Guide, Version 1.2*, Westford, USA, 2000.

[Horwitz95] Horwitz, P., *Linking Models to Data: Hypermodels for Science Education*, The High School Journal 79(2):148-156, 1995.

[IEEE02] IEEE Learning Technology Standards Committee (LTSC), Learning Object Metadata Working Group, *Draft Standard for Learning Object Metadata (IEEE 1484.12.1-2002)*, July 2002.

URL: <http://ltsc.ieee.org/wg12>

[IMS01] IMS Global Learning Consortium, Inc., *IMS Content Packaging Specification, Version 1.1.2*, August 2001.

URL: <http://www.imsproject.org/content/packaging>

[Jaspers91] Jaspers, F., *Interactivity or instruction? A reaction to Merrill*, Educational Technology 31(3):21-24, 1991.

[Johnson98] Johnson, N. F., Jajodia, S., *Exploring Steganography: Seeing the Unseen*, IEEE Computer, 31(2):26-34, February 1998.

- [Ingalls81] Ingalls, D. H. H., *Design Principles Behind Smalltalk*, BYTE Magazine, August 1981.  
URL: [http://users.ipa.net/~dwright/smalltalk/byte\\_aug81/design\\_principles\\_behind\\_smalltalk.html](http://users.ipa.net/~dwright/smalltalk/byte_aug81/design_principles_behind_smalltalk.html)
- [Johnson89] Johnson, J., Roberts, T. L., Verplank, W., Smith, D. C., Irby, C. H., Beard, M., Mackey, K., *The Xerox Star: A retrospective*, IEEE Computer, 22(9):11-29, 1989.  
URL: <http://www.digibarn.com/friends/curbow/star/retrospect>  
URL: <http://www.digibarn.com/stories/desktop-history/bushytree.html>  
The updated Bushy tree by John Redand and Bruce Damer.
- [Kay77] Kay, A., Goldberg, A., *Personal Dynamic Media*. IEEE Computer, 10(3):31-41, March 1977.
- [Kay69] Kay, A., *The Reactive Engine*, Ph.D. thesis, University of Utah, 1969.  
URL: <http://www.mprove.de/diplom/gui/kay69.html>
- [Kay91] Kay, A., *Computers, Networks & Education*, Scientific American Magazine, September 1991.
- [Kay93] Kay, A., *The early history of Smalltalk*, History of Programming Languages archive, ACM SIGPLAN Notices, 28(3):69-95, 1993.
- [Kearsley03] Kearsley, G., *Explorations in Learning & Instruction: The Theory Into Practice Database*, August 2003.  
URL: <http://tip.psychology.org>
- [Klein98a] Klein, R., Hanisch, F., Straßer, W., *Web- based Teaching of Computer Graphics: Concepts and Realization of an Interactive Online Course*, in: Michael Cohen (eds), SIGGRAPH 98 Conference Proceedings, Addison Wesley, 1998.
- [Klein98b] Klein, R., Hanisch, F., *Using a modular construction kit for the realization of an interactive Computer Graphics course*, in: Proc. of ED-MEDIA & ED-TELECOM, AACE, USA, 1998.  
URL: <http://www.gris.uni-tuebingen.de/projects/grdev>  
Computer Graphics courseware
- [Knox99] Knox, D., Fincher, S., Dale, N., Adams, E., Goelman, D., Hightower, J., Loose, K., Springsteel, F., *The Peer Review Process of Teaching Materials, Report of the ITiCSE'99 Working Group on Validation of the quality of teaching materials*, ITiCSE'99 Working Group Reports 31(4), 1999.
- [Kortenkamp99] Kortenkamp, U., *Foundations of Dynamic Geometry*, Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, 1999.  
URL: <http://www.cinderella.de/papers/diss.pdf>
- [Kortenkamp02] Kortenkamp, U., Richter-Gebert, J., *Making the Move: The Next Version of Cinderella*, Proc. of the First International Congress Of Mathematical Software, World Scientific, pp.208ff., 2002.

- [Krasner88] Krasner, G. E., Pope, S.T., *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 system*, Journal of Object Oriented Programming, 1(3):26-49, 1988.
- [Kuisma02] Kuisma, J. J., Watson, J. A., *Scheduling and Monitoring Dynamic Learning Objects on the Web*, Proc. of the ED-MEDIA 2002 Conference, Denver, CO, June 2002.
- [Kynigos01] Kynigos, C., Friedmann, J., *E-Slate Profile*, in: diSessa, A. (ed.), the Web/comp project, March 2001.  
URL: <http://www.soe.berkeley.edu/~boxer/webcomp/>
- [Laleuf01] Laleuf, J. R. Spalter, A. M., *A Component Repository for Learning Objects: A Progress Report*, Proceedings of ACM JCDL2001, Roanoke, VA, 2001.  
URL: <http://www.cs.brown.edu/research/graphics/research/exploratory>  
The Exploratories Project
- [Lacroute94] Lacroute, p. , Levoy, M., *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*, Proc. SIGGRAPH '94, pp.451-458, Orlando, Florida, July, 1994.
- [LSAL03] Learning Systems Architecture Lab, *SCORM Best Practices Guide for Content Developers*, 1<sup>st</sup> edition, Carnegie Mellon University, USA, February 2003.  
URL: <http://www.lsal.cmu.edu/lsal/expertise/projects/developersguide>
- [Licklider60] Licklider, J. C. R., *Man-Computer Symbiosis*, IRE Transactions on Human Factors in Electronics, 1:4-11, March 1960.  
URL: <http://memex.org/licklider.pdf>
- [Löringhoff78] v. Freytag Löringhoff, B.B., *Die Rechenmaschine*, in Seck, F., Mohr, J. C. B. (eds.) Wilhelm Schickard, Paul Siebeck, Tübingen 1978.
- [MathForum03] The Math Forum, *Math Forum @ Drexel*, Drexel University, 1994-2003.  
URL: <http://mathforum.org>
- [Maurer96] Maurer, H. (ed.), *Hyperwave: The Next Generation Web Solution*, Addison-Wesley Longman, London, 1996.  
URL: <http://www.iicm.edu/hwbook>
- [Maurer97] Maurer, H., *Necessary Ingredients of Integrated Network Based Learning Environments*, Proc. of ED-MEDIA 97 – World Conference on Educational Multimedia and Hypermedia, pp.709-716, AACE, June 1997.
- [Maurer01a] Maurer, H., Sapper, M., *E-Learning Has to be Seen as Part of General Knowledge Management*, Proc. of ED-MEDIA 2001, pp.1249-1253, AACE, Charlottesville, USA, 2001.  
URL: [http://www.iicm.edu/iicm\\_papers/e-learning\\_part\\_of\\_KM.doc](http://www.iicm.edu/iicm_papers/e-learning_part_of_KM.doc)
- [Maurer01b] Maurer, H., *Computer-Based Teaching/Web-Based Teaching*, in: Rojas, R. (ed.), Encyclopedia of Computers and Computer History, 1:181-182, Fitzroy Dearborn Publishers, Chicago, 2001.



URL: [http://www.iicm.edu/iicm\\_papers/cbt\\_wbt.doc](http://www.iicm.edu/iicm_papers/cbt_wbt.doc)

- [Maurer02] Maurer, H., *What have we learnt in 15 years about educational multimedia?*, Keynote Presentation Ed-Media 2002, in: Proc. ED-MEDIA 2002, 1:2-7, AACE Charlottesville, USA, 2002.

URL: [http://www.iicm.edu/iicm\\_papers/edmedia\\_2002.doc](http://www.iicm.edu/iicm_papers/edmedia_2002.doc)

- [Meyrowitz86] Meyrowitz, N., *Intermedia: The architecture and construction of an object-oriented hypemedia system and applications framework*, Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications, pp.186-201, Portland, OR, ACM, September 1986.

- [Meyrowitz87] Meyrowitz, N., *The Missing Link: Why We're All Doing Hypertext Wrong*, in: *The society of text: Hypertext, hypermedia and the social construction of information*, pp.107-114, MIT Press, 1989.

- [Meyrowitz89] Meyrowitz, N., *The Desktop of Tomorrow: From User-Centered to Information-Centered Computing*, Draft 0.95, Institute for Research in Information and Scholarship (IRIS), Brown University, Providence, July 1989.

URL: <http://klynch.com/documents/tomorrow>

- [Merrill80] Merrill, M. D., *Learner control in computer based learning*, Computers and Education, 4:77-95, 1980.

- [MuellerProve02] Müller-Prove, M., *Vision and Reality of Hypertext and Graphical User Interfaces*, Department of Informatics, University of Hamburg, February 2002.

- [Myers98] Myers, B. A., *A Brief History of Human Computer Interaction Technology*, ACM interactions, 5(2):44-54, March 1998.

- [Nelson65] Nelson, T. H., *A File Structure for the Complex, the Changing and the indeterminate*, Proc. ACM National Conference. pp. 84-100, 1965.

- [Nelson82] Nelson, T. H., *Literary Machines*, Mindful Press, Sausalito, California, 1982.

URL: <http://xanadu.com>

- [Nelson99] Nelson, T. H., *Xanalogical Structure, Needed Now More than Ever: Parallel Documents, Deep Links to Content, Deep Versioning and Deep Re-Use*, ACM Computing Surveys 31(4), December 1999.

- [Nielsen95] Nielsen, J., *Multimedia and Hypertext: The Internet and Beyond*, AP Professional, Boston, 1995.

- [NSF03] National Science Foundation, National Science, *Technology, Engineering, and Mathematics Education Digital Library (NSDL)*, National STEM Education Digital Library Program, Division of Undergraduate Education, NSF, 2003.

URL: <http://www.ehr.nsf.gov/duel/programs/nsdl/>

- [Owen00] Owen, S. G., Sunderraman, R., Zhang, Y., *The development of a digital library to support the teaching of computer graphics and visualization*. Computer & Graphics 24(4):623-627, 2000.

- [Papert80] Papert S., *Mindstorms: Children, Computers and Powerful Ideas*, Basic Books, Inc., New York, 1980.
- [Piaget70] Piaget, J., *The Science of Education and the Psychology of the Child*, Grossman, New York, 1970.
- [Piaget73] Piaget, J., *To Understand Is To Invent*, The Viking Press, Inc, New York, 1972.  
URL: [www.montclair.edu/crc/piaget.html](http://www.montclair.edu/crc/piaget.html)
- [Parnafes01] Parnafes, O., diSessa, A., *ESCOT Profile*, in diSessa, A. (ed.), the Web/comp project, March 2001.  
URL: <http://www.soe.berkeley.edu/~boxer/webcomp/>
- [Repenning95] Repenning, A., T. Sumner, *Agentsheets: A Medium for Creating Domain-Oriented Visual Languages*, IEEE Computer, 28:17-25, 1995.
- [Repenning01] Repenning, A., Ioannidou, A., Payton, M., Ye, W., Roschelle, J., *Using Components for Rapid Distributed Software-Development*, IEEE Software, 18(2):38-45, 2001.  
URL: <http://www.escot.org>  
Educational Software Components of Tomorrow
- [Resnick91] Resnick, M., Ocko, S., *LEGO/Logo: Learning Through and About Design*, in Harel I., Papert, S. (eds.), *Constructionism*, pp.141-150, Ablex Publishing, Norwood, NJ, 1991.
- [Rhodes85] Rhodes, D.M., Azbell, J.W., *Designing interactive video instruction professionally*, Training and Development Journal, 39(12):31-33, 1985.
- [Rogers98] Rogers, Y., Scaife, M., *How can interactive multimedia facilitate learning?*, in Lee, J. (ed.), *Intelligence and Multimodality in Multimedia Interfaces: Research and Applications*, AAAI Press, Menlo Park, CA, 1998.  
URL: <http://www.cogs.susx.ac.uk/users/yvonner/ecoihome/IMMI.html>
- [Romiszowski86] Romiszowski, A. J., *Developing auto-instructional materials*, Michols Publishing Company, New York, 1986.
- [Roschelle96] Roschelle, J., Kaput, J., DeLaura, R., *Scriptable applications: Implementing open architectures in learning technology*, in: Carlson, p. and Makedon, F. (eds.), *Proc. of Ed-Media 96 – World Conference on Educational Multimedia and Hypermedia*, pp.599–604, AACE, 1996.
- [Roschelle98] Roschelle, Jeremy, Jim Kaput, Walter Stroup, Ted M. Kahn. *Scaleable Integration of Educational Software: Exploring the Promise of Component Architectures*, Journal of Interactive Media in Education 98, Oct. 1998.  
URL: <http://www-jime.open.ac.uk/98/6>
- [Roschelle99] Roschelle, J., DiGiano, C., Koutlis, M., Repenning, A., Jackiw, N., Suthers, D, *Developing educational software components*. IEEE Computer, 32(9):50-58, September 1999.



- [Schmaranz96] Schmaranz, K., *Professional Electronic Publishing in Hyper-G: The Next Generation Publishing Solution on the Web*, Journal of Universal Computer Science 2(9):650-658, Springer Pub. Co., Graz University of Technology, Austria, September 1996.
- [Schulmeister97] Schulmeister, R., *Hypermedia Learning Systems: Theory – Didactics – Design*. English online version of "Grundlagen hypermedialer Lernsysteme", 2nd ed., Oldenbourg, München, 1997.  
URL: [http://www.izhd.uni-hamburg.de/paginae/Book/Frames/Start\\_FRAME.html](http://www.izhd.uni-hamburg.de/paginae/Book/Frames/Start_FRAME.html)
- [Schulmeister03] Schulmeister, R., *Lernplattformen für das virtuelle Lernen: Evaluation und Didaktik*. Oldenbourg, München, 2003.
- [Schwier93] Schwier, R.A., Misanchuk, E., *Interactive Multimedia Instruction*, Englewood Cliffs, NJ: Educational Technology Publications, Inc., 1993.
- [Shneiderman82] Shneiderman, Ben (ed.), *Designing the User Interface*. Addison-Wesley, Reading, MA, 1992.
- [Shneiderman89] Shneiderman, B., *Reflections on Authoring, Editing and Managing Hypertext*, in: The society of text: Hypertext, hypermedia and the social construction of information}, pp.115?-131, MIT Press, 1989.
- [Shneiderman97] Shneiderman, Ben, *Direct manipulation for comprehensible, predictable, and controllable user interfaces*, Proc. of the ACM International Workshop on Intelligent User Interfaces '97, pp.33-39, New York, 1997.
- [Skinner54] Skinner, B. F., *The science of learning and the art of teaching*, Harvard Educational Review, 24(2):86-97, 1954.
- [Simpson99] Simpson R. M., Spalter A. M., van Dam A., *Exploratories: An Educational Strategy for the 21st Century*, Proc. of ACM SIGCSE 1999, 1999.
- [Sims95] Sims, R., *Interactivity: A Forgotten Art?* ITFORUM, Paper #10, November 1995.  
URL: <http://it.coe.uga.edu/itforum/paper10/paper10.html>
- [Sims00] Sims, R., *An interactive conundrum: Constructs of interactivity and learning theory*, Australian Journal of Educational Technology, 16(1):45-57, 2000.  
URL: <http://www.ascilite.org.au/ajet/ajet16/sims.html>
- [Singh02] Singh, I., Stearns, B., Johnson, M., *Designing Enterprise Applications with the J2EETM Platform*, Second Edition. Addison-Wesley, 2001.  
URL: <http://java.sun.com/blueprints/patterns/MVC-detailed.html>
- [Smith77] Smith, D. C., *Pygmalion: A Computer Program to Model and Simulate Creative Thought*, Birkhauser Verlag, Basel and Stuttgart, 1977.  
URL: <http://www.acypher.com/wwid/Chapters/01Pygmalion.html>  
A commentary of Smith on Pygmalion

- [Smythe02] Smythe, C., Shepherd, E., Brewer, L., Lay S., *IMS Question & Test Interoperability: ASI Information Model Specification*, Final Specification, Version 1.2, IMS Global Learning Consortium, Inc., February 2002.  
URL: <http://www.imsproject.org/question>
- [SmithGratto02] Smith-Gratto, K., Wicks, D., Berger, C., *MERLOT: Reaping the On-line Vineyard*, Proc. ED-MEDIA 2002, 1:2-7, AACE, Charlottesville, USA, 2002.  
URL: <http://www.merlot.org>  
Multimedia Educational Resource for Learning and Online Teaching
- [Spalter00] Spalter, A. M., Simpson R., M., *Reusable Hypertext Structures for Distance and JIT Learning*, Proc. of ACM Hypertext '00, 2000.
- [Spalter03] Spalter, A. M, van Dam, A., *Problems with Using Components in Educational Software*, Computer & Graphics 27(3):329-337, Elsevier Science Ltd., June 2003.
- [Strauss92] Strauss, p. S., Carey, R., *An object-oriented 3D graphics toolkit*, Computer Graphics Siggraph 92, pp.341-349, ACM Siggraph, July 1992.
- [Sun98] Sun Microsystems, Inc., *Proposal for a Drag and Drop subsystem for the Java Foundation Classes*. Final Draft v0.96, August 1998.  
URL: <http://www.java.sun.com/products/javabeans/glasgow>
- [Sun03] Sun Microsystems, Inc., *Java 2 Platform, Standard Edition (J2SE)*, 1995-2003.  
URL: <http://java.sun.com/j2se>
- [Sutherland63] Sutherland, I. E., *Sketchpad – A Man-Machine Graphical Communication System*, AFIPS Spring Joint Computer Conference (SJCC '63), pp.329-346, 1963.
- [Tesler81] Tesler, L., *The Smalltalk Environment*, Byte, 6(8), pp.90-147, August 1981.
- [Thorndike22] Thorndike, E., *The Psychology of Arithmetic*. Macmillan, New York, 1922.
- [Tognazzini03] Tognazzini, B., *First Principles*, in: ASKTOG, May, 2003.  
URL: <http://asktog.com/basics/firstPrinciples.html>
- [vanDam69] van Dam, A., Carmody, S., Gross, W., Nelson, T. H., Rice, D., *A Hypertext Editing System for the /360*, Proc. of the Second University of Illinois Conference on Computer Graphics, University of Illinois, 1969.
- [vanDam87] van Dam, A., *Hypertext '87 Keynote Address*, Communications of the ACM 31(7):887-895, 1988.  
URL: <http://www.cs.brown.edu/memex>
- [vanDam97] van Dam, A., *Post-WIMP user interfaces*, Communications of the ACM, 40(2):63-67, 1997.

- [vanDam02] van Dam, A., *Next-Generation Educational Software*, Brown University and the NSF STC for Graphics and Visualization, ED-MEDIA 2002 Keynote Presentation, Denver, Colorado, June 2002.  
URL: <http://www.cs.brown.edu/people/avd>
- [Weber97] Weber, G., Specht, M., *User modeling and adaptive navigation support in WWW-based tutoring systems*. Proc. of User Modeling '97, pp.289-300, 1997.
- [W3C03] World Wide Web Consortium (W3C), *W3C Technical Reports and Publications*, W3C, 1994-2003.  
URL: <http://www.w3.org/TR>
- [Whitehead98] Whitehead, E. J., JR., Wiggins, M., *WebDAV: IETF Standard for Collaborative Authoring on the Web*, IEEE Internet Computing 2(5):34-40, 1998.
- [Wiest01] Wiest, S., Zell, A., *Improving Web Based Training Using an XML Content Base*, in: Kalpic, D., Dobric, V., (eds.), Proc. of the 22nd Intl. Conf. Information Technology Interfaces ITI 2000, pp.229-234, Pula, Croatia, June, 2000.
- [Wiley00] Wiley, D. A. (ed.), *The Instructional Use of Learning Objects*, Agency for Instructional Technology, January 2002.  
URL: <http://www.reusability.org/read>
- [Yaron01] Yaron, D., Milton, D. J., Freeland, R., *Linked active content: A service for digital libraries for education*, ACM Proceedings of the Joint Conference on Digital Libraries, pp.25-32, 2001.
- [Yaron02] Yaron, D., Milton, J., Freeland, R., *Linked Active Content for Digital Libraries for Education*, Journal of Digital information 2(4), 2002.  
URL: <http://jodi.ecs.soton.ac.uk/Articles/v02/i04/Yaron>