The AudioGraph Web-based Lecturing System

A dissertation presented in partial

Fulfilment of the requirements for the degree of

Doctor of Philosophy

In

Computer Science

At Massey University, Palmerston North,

New Zealand.

Horia Cristian Slusanschi

2004

Abstract

In recent years, the pace of technological development has grown tremendously. As a result, the half-life of knowledge has decreased from hundreds and tens of years to just a few years or even mere months in certain fields. In order to be able to adapt efficiently to this unprecedented wave of knowledge, organizations and individuals must adopt new ways of learning and training.

Web-based education is a viable solution to the problem of quickly disseminating fresh knowledge as it emerges. However, one of the main challenges of effective web-based education remains the application of sound educational principles in the design and delivery of technology-enabled courses.

Our primary aim has been to provide lecturers with an effective and easy to use system that would assist web-based teaching and learning, allowing them to focus on the application of relevant educational principles rather than requiring them to master arcane technical complexities.

To enhance our perspective and better inform our design decisions, we explore the factors contributing to effective web-based education and examine a number of existing lecturing systems applicable to the development and distribution of educational content over the Web.

We then investigate means of preparing and presenting educational content. We place particular emphasis on the guiding principles of human interface design that have permeated our work, and have contributed to the enhanced usability characteristics of our system. We then discuss the challenges of data compression and review the technologies we chose to help make our approach viable and efficient.

In describing the software architecture of the system, we introduce the various design patterns that have helped our successful implementation gain robustness and flexibility, and discuss various tradeoffs encountered throughout the design and development of the system.

Finally, we present the conceptual and technical conclusions we have reached and we briefly explore future avenues of research and indicate a number of particularly interesting and potentially fruitful areas.

Acknowledgements

I would like to acknowledge a number of people without which this endeavour would not have been possible.

A special thank you must go to Professor Chris Jesshope, my supervisor, for all the trust he has put in my efforts, and especially for making it all possible. I would also like to extend my deepest gratitude towards Dr. Eva Heinrich for her relentless excellent suggestions and gracious guidance of my later work of improving the presentation of this research with invaluable advice.

I would like to thank the many people at Massey University, which I have met in the past few years, whose names are too many to mention here, and who have already put a substantial amount of effort into using and further developing the AudioGraph system, for their tireless persistence and innumerable excellent suggestions.

Most importantly, I would like to thank my wife, Anca, for her constant encouragement and enduring patience throughout the duration of this work.

Table of Contents

Table of Figures

Massey University
COLLEGE OF SCIENCES

Institute of Information Sciences & Technology
Private Bag 11 222,
Palmerston North,
New Zealand
Telephone: 64 6 350 5799
Facsimile: 64 6 350 5723

Integrated Research and Teaching in the Fields of:

Statistics & Applied Statistics

Computer Science & Information Systems

Information & Electronic Engineering

Candidate's Declaration

This is to certify that the research carried out for my Doctoral Thesis entitled "The AudioGraph Web-based Lecturing System" in the Institute of Information Sciences and Technology, Massey University, Palmerston North Campus, New Zealand is my own work and that the thesis material has not been used in part or in whole for any other qualification.

Candidate's Name: Horia Cristian Slusanschi

Signature: 

Date: 09/02/2004

Massey University
COLLEGE OF SCIENCES

Institute of Information Sciences & Technology
Private Bag 11 222,
Palmerston North,
New Zealand
Telephone: 64 6 350 5799
Facsimile: 64 6 350 5723

Integrated Research and Teaching in the Fields of:

Statistics & Applied Statistics

Computer Science & Information Systems

Information & Electronic Engineering

Supervisor's Declaration

This is to certify that the research carried out for the Doctoral Thesis entitled "The AudioGraph Web-based Lecturing System" was done by Horia Cristian Slusanschi in the Institute of Information Sciences and Technology, Massey University, Palmerston North Campus, New Zealand. The thesis material has not been used in part or in whole for any other qualification, and I confirm that the candidate has pursued the course of study in accordance with the requirements of the Massey University regulations.

Supervisor's Name: Dr. Eva Heinrich

Signature: 

Date: 04/02/2004

Massey University
COLLEGE OF SCIENCES

Institute of Information Sciences & Technology
Private Bag 11 222,
Palmerston North,
New Zealand
Telephone: 64 6 350 5799
Facsimile: 64 6 350 5723

Integrated Research and Teaching in the Fields of:

Statistics & Applied Statistics

Computer Science & Information Systems

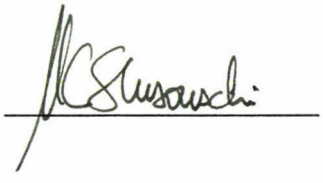
Information & Electronic Engineering

Certificate of Regulatory Compliance

This is to certify that the research carried out in the Doctoral Thesis entitled "The AudioGraph Web-based Lecturing System" in the Institute of Information Sciences and Technology, Massey University, Palmerston North Campus, New Zealand:

- Is the original work of the candidate, except as indicated by appropriate attribution in the text and/or in the acknowledgements;
- That the text, excluding appendices/annexes, does not exceed 100,000 words;
- All the ethical requirements applicable to this study have been complied with as required by Massey University and relevant legislation.

Candidate's Name: Horia Cristian Slusanschi Supervisor's Name: Dr. Eva Heinrich

Signature:  Signature: 

Date: 09/02/2004 Date: 04/02/2004

1 Introduction

Web-based education has enjoyed a rapidly increasing amount of attention from academia and industry alike in recent years, and it forms the context of the research presented in this paper.

In this chapter, we will show the motivating forces behind our work and indicate our specific area of interest — web-based lecturing. To provide a wider context, we will then briefly introduce the main factors that contribute to effective web-based education.

We will then introduce the AudioGraph idea and show its vision and intended audience. To clarify the context of our work, we introduce the precursor AudioGraph prototype and examine its valuable concepts and its drawbacks.

From this background we derive our research objectives, describe the methodology we envisaged for our work and show how it is reflected in the structure of this document.

1.1 Motivating Forces

In the course of history, communication between humans has taken place through various media, as facilitated by the technology of the time. Since education takes place through the communication of information from generation to generation, the history of communication methods and technologies could also be seen as the history of education media.

For instance, in prehistoric times, it is widely believed that human communication has evolved through sign language, and then gradually to spoken language. However, language on its own has no intrinsic persistence, and human memory has not proven to be a very reliable long-term repository for information, with very rare exceptions.

The earliest available Egyptian hieroglyphs are dated at least around the XXXIV-th century B.C. [263], and they have since been used to record information in ivory labels and tablets, stone and later on papyrus paper. Along the course of history, Egyptians seem to have used writing for various religious, domestic and commercial purposes.

The Egyptian writing systems have influenced most scripts that would later emerge in Europe and the Middle East. The hieroglyphic symbols were used for their sound values, as a form of syllabic alphabet. This system of writing was developed in three successive stages, known as hieroglyphic, hieratic, and demotic. The oldest form, hieroglyphic, dealt primarily with sacred texts and symbols. The hieratic script came later on into being as a simplified, cursive script used by the intellectuals, while the demotic script was derived as an even further simplification into a popular cursive script around the VII-th century B.C.

Current historical data shows that the Sumerians started to use writing some time between the XXXIII-rd and the XXX-th century B.C. and recorded their writings primarily on clay tablets [264]. The Sumerians appear to have been compulsive writers and recorded most aspects of their lives and history on the clay tablets. As a result, only a fraction of the total number of tablets found by archaeologists has been deciphered to date. Current efforts are underway to record such ancient tablets in holographic form so that more scholars can examine and work on deciphering them. Babylonians and Assyrians have developed the cuneiform script — deriving it from the Sumerian script — and have recorded vast amounts of information on clay tablets. The Sumerian and cuneiform scripts have developed and vanished in Mesopotamia.

Writing systems can largely be classified as ideographic — with symbols representing concepts or words — syllabic and alphabetic — with symbols denoting sound values to aid in the formation of words that would convey concepts. Records of alphabetic scripts have appeared almost a millennia and a half after the first available Egyptian and Sumerian records.

To gain an appreciation of the extraordinary variety of the means of communication devised by humans, it is worth noting that the South American Inca civilization has used coloured knotted strings — called *quipu* — to record information and communicate it using an elaborate mathematical code [265].

With the development of alphabets and writing on parchment, vast libraries were soon formed, as the legendary library of Alexandria. The administration of the Roman Empire has used vast quantities of parchment for keeping detailed administrative records, which were a key factor in establishing and maintaining the stability of the Empire. The Chinese have invented and then used paper for writing for at least two and a half millennia.

Although Chinese engravers had perfected the art of woodblock printing since at least the VI-th century A.D., it took Guttenberg's moveable type printing revolution in the XV-th century to accelerate significantly the spread of information and knowledge. The technological advances made mass production of written materials possible, and therefore information could be distributed much faster, fuelling the accelerated development of science and new ideas. The first newspapers appear throughout the XVII-th and early XVIII-th centuries.

Later on, in the XIX-th century, photography is developed, and photographs begin to be used in conjunction with written text to enrich the information being communicated. Previously, texts have been annotated with diagrams and drawings for many centuries.

The development of the telegraph, telephone and phonograph in the XIX-th century have then made possible the even quicker distribution of information and synchronous communication through voice over very large distances, and the recording and reproduction of sound.

Soon afterward, further technological advances brought motion pictures, radio and television into the mainstream culture throughout the developed world, facilitating ever-richer communication among humans.

The spectacular development of electronics and computers in recent times has revolutionised the way information is recorded, stored and transmitted. With the widespread development of the World-Wide-Web, information can be shared virtually instantaneously by people across the globe. In the search for ever-better communication, education and entertainment solutions, the newly revealed flexibility and capability of computers have also made possible the development of new, interactive and dynamic media.

The vast technological and scientific advances of the last century have led to an unprecedented increase in humankind's total body of knowledge, and the pace of change continues to accelerate. Currently, there does not seem to be any practical limit to this accelerating advance of knowledge and technology apart from our ability to grasp its implications and make wise and efficient use of it.

Since our traditional educational systems have developed through the ages around a nearly static body of knowledge, they are starting to lose a good deal of their effectiveness, as in some cases the information traditional courses are based on becomes obsolete in a matter of months.

Therefore, humankind's approach to education needs to evolve as well in order to cope with such rapid changes. Not surprisingly then, given that it offers promising results in mitigating the limiting factors in traditional educational approaches, the field of web-based education has witnessed an explosive growth over the past few years, and a number of factors are continuing to accelerate this process.

Kurzweil presents an interesting vision of the future, suggesting that within the next ten years intelligent courseware may become the common means of learning ([262] pp. 240). He also proposes that web-based learning will become commonplace and that learning itself will become a significant portion of most jobs, as an ongoing responsibility, not just an occasional supplement. Current experience suggests that his proposals seem to match recent developments in our society.

Technology has reshaped the way business is being conducted. Many businesses find that in order to develop or maintain their competitive edge, they must ensure that their workforce takes advantage of the latest technological developments. Failure to do so only means that competitors will seek the same advantage, and as a result, we are witnessing a race of technological development and adoption.

The people that make businesses work must keep pace with the ever changing sets of technical and business skills required of them. Consequently, they must be frequently trained and assisted

in their development of new skills. The traditional educational patterns, where students concentrate full-time for a few years on various subjects in order to obtain a degree are no longer the best fit for these educational demands.

As the Internet has started to become a commodity, the technology required to disseminate information and knowledge has become less and less expensive. However, although the means to distribute information have become increasingly accessible, the systems required to shape, manage, and present information in various forms are still being developed and refined.

Educational paradigms that suit current training requirements need to be considerably more flexible, instantly available if possible, and reach students almost everywhere, distracting them from their business responsibilities as little as possible.

In this climate, there is a great need for adequate systems to create, manage and distribute good quality educational and informational content over the web. Web-based education, by taking advantage of the Internet as a distribution medium with rapid and very wide reach, promises to address these issues and provide a viable solution.

A central component of web-based education is the design and delivery of teaching material. Such material must be designed according to sound educational principles, by taking into account all the factors of effective web-based education, as shown in Figure 1.1.

The research described here explores the issues involved in developing an efficient web-based multimedia lecture authoring system for creating, managing and distributing information to support education, using the Internet as a dissemination medium, in a bid to facilitate the enhancement of traditional education.

1.2 Web-based Education

In the accelerating modern technological evolution, ongoing education has become an essential part of most careers. Traditional educational institutions, however, were not designed to cope with such rapidly escalating educational demands. Therefore, in order to meet the growing learning needs of ever increasing numbers of students, new approaches must be found and old approaches must be adapted and upgraded to meet the current challenges.

Web-based education is one approach that shows good promise for addressing the current educational crisis. Fundamentally, web-based education relies on online materials and learning resources, such as text and multimedia lectures, tutorials, exercises, quizzes, tests, etc. It can also make use of computer-mediated communication, facilitating online interactivity and collaboration through peer-to-peer and peer-to-tutor communication.

Web-based education promises to deliver a number of key benefits:

- It may be possible to maintain or even enhance the effectiveness of education while educational costs are reduced by taking advantage of the more cost-efficient information dissemination mechanisms provided by the web for wider audiences in contrast with traditional educational approaches.
- The most up-to-date information can reach students in the shortest possible time, as it is quickly distributed over the Internet. They can then immediately apply, test and further develop their knowledge, which in turn, in a virtuous circle, may then sustain the accelerated pace of scientific and technological advancement.
- Due to increased use of web-based education solutions, teachers may gain precious time, as they are freed up from the mostly repetitive task of presenting classes. Teachers can start to focus on clarifying complex issues and "attend primarily to issues of motivation, psychological well being, and socialisation", as suggested by Kurzweil [262] (pp. 241). Teachers may also focus more of their efforts on research and the general advancement of science.
- Students gain more freedom in developing their learning habits and choosing their own approach to education. Each student can progress through the course material at her own pace, thus freeing gifted students to cover more ground and broaden their horizon faster, and allowing other students to focus on complex issues at a rate that ensures comprehension for them.

In order to reap these benefits, our whole approach to learning and education must evolve. The traditional educational environment and its associated institutions must be transformed to provide support for the emerging learning paradigms.

For the successful, widespread adoption and further development of this educational paradigm, there are a number of challenges to overcome.

As institutions have always had a considerable amount of inertia, some time will be required until the shift in the traditional educational patterns will be reflected in the operation of educational institutions with traditional backgrounds.

There are also cultural limitations as well, since what is readily accepted in one culture does not necessarily receive the same adoption in another. Then there are technical challenges, such as authoring and adapting suitable educational content, bandwidth limitations and electronic data management complexities to overcome.

Figure 1.1 shows the main factors that must be considered in concert in order to achieve effective web-based education, such as authoring tools for online lectures, appropriate

educational approaches, teaching and learning styles, instructional design methodologies and practices and learning management systems.

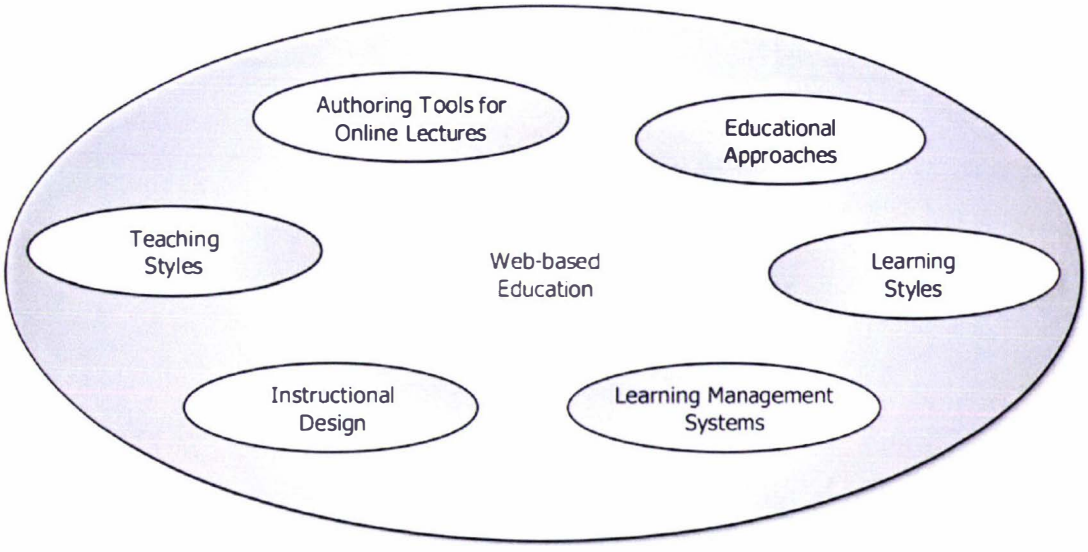


Figure 1.1 Factors contributing to effective web-based education

Many organizations have sought to build their web-based education solutions on existing materials, while others have chosen to investigate new means of developing and distributing educational content as well in order to provide a more compelling experience for their students. To achieve this added dynamism, new authoring paradigms, means of expression and supporting tools had to be devised.

One of the most severe drawbacks of such new authoring systems is generally a steep learning curve, as complex functions may be involved in building media-rich content. A possible answer to this deficiency is to develop standardised applications that over time become familiar to a large number of teachers and content authors so that the learning curve becomes shallow due to the common knowledge.

In order to capture the multiple dimensions of face-to-face lectures as best as possible and provide the best conditions for achieving the desired learning outcomes, authoring tools for online lectures should use a multimedia format. Systems for the development of such multimedia teaching material must be easy to use, in order to be accessible to virtually all educators. Similarly, to be accessible to as many students as possible, it should be possible to distribute the lecture material over the Internet with great bandwidth economy.

Maurer suggests that although the progress in handling media has helped improve the quality of technology-assisted education, the major breakthroughs have been on other fronts, such as the better understanding of different learning models or educational approaches — behaviourism, cognitivism and constructivism or incidental versus explicit learning [26].

Modern educational approaches cover a spectrum between the following three broad groups: behaviourist, cognitivist, and constructivist [27]. While the behaviourist theory focuses primarily on the role and impact of the lecturer, the constructivist theory is centred on the student, with cognitivist theory situated somewhere in between these two perspectives.

According to behaviourist theory, learning is manifested through changes in either the form or frequency of observable performance [29]. Learning can be demonstrated through appropriate responses as a result of specific environmental stimuli.

Cognitivist theory stresses the acquisition of knowledge and the mental processes involved in this process. Its affinities lie within the rationalist rather than the empiricist end of the epistemology continuum.

Constructivist theory lies at the confluence of rationalist — regarding the mind as the source of all meaning — and empiricist currents — considering the direct interaction with the environment crucial. An interesting characteristic of this approach — in contrast with the cognitivist and behaviourist theories — lies in the opinion that knowledge cannot be considered to be fully independent of the student's psyche and experience.

In other words, meaning can only be created through personal experience rather than be acquired in an objective form. Although the existence of an objective reality is not denied, constructivist theory argues that profound knowledge arises fundamentally from personal experience and interpretation[1].

Maurer reports that it is now widely recognised that the behaviouristic approach is seen to be most appropriate for rote learning of well-established facts, while cognitivistic approaches — requiring real, deep contextual understanding — or constructivistic techniques — where learners have to actually discover, not just to understand situations and solutions — are generally more desirable in modern education, given the rapid flux and evolution of contemporary culture and knowledge [26].

Given the current relevance and prevalence of cognitivistic and constructivistic approaches, and due to their perspective on learning and knowledge development as mental activities, consideration of the main teaching and learning styles may yield valuable insights on the influences lecturers are likely to impart to students, and the approaches students may take to assimilate knowledge.

---

[1] In certain ways, this insight of constructivist thinking could be seen to mirror the evolution of modern physics from a Newtonian, fully deterministic perspective (which might be likened to behaviouristic views), through relativistic approaches (which might be interpreted as cognitivistic in nature due to their more profound underpinnings), towards the quantum theories, which highlight the fact that at certain scales, no matter how carefully constructed an experiment might be, the process of observation affects the process or entity being observed, effectively concluding that even theoretically no two perfectly identical measurements can ever be made. Similarly, constructivist theory suggests that no two students are ever identical; the specific history of learning experiences that they have lived through leaves a unique imprint in each of them, and their interpretations of the current situations are constantly influenced by it.

Grasha and Richlin identified the following five teaching styles in describing the prevalent aspects of faculty presence in the classroom: *expert*, *formal authority*, *personal model*, *facilitator* and *delegator* [45].

The *experts* possess knowledge and expertise that students need. They strive to maintain status as experts among students by displaying detailed knowledge and by challenging students to enhance their competence. They are concerned with transmitting information and insuring that students are well prepared.

Formal authority exponents possess status among students due to their knowledge and role as a faculty member. They are concerned with providing positive and negative feedback, establishing learning goals, expectations, and rules of conduct for students. They are also concerned with the correct, acceptable, and standard ways to do things and with providing students with the structure they need to learn.

Representatives of the *personal model* style believe in *teaching by personal example* and establish a prototype for how to think and behave. They oversee, guide and direct by showing how to do things, and encourage students to observe and then to emulate the instructor's approach.

Facilitators emphasise the personal nature of teacher-student interactions. They guide and direct students by asking questions, exploring options, suggesting alternatives and encouraging them to develop criteria to make informed choices. Their overall goal is to develop in students the capacity for independent action, initiative, and responsibility. They work with students on projects in a consultative fashion and try to provide as much support and encouragement as possible.

The *delegators* are primarily concerned with developing students' capacity to function in an autonomous fashion. Students work independently on projects or as part of autonomous teams. The teacher is available at the request of students as a resource person.

Also, the study of learning styles may help us to better understand students and their learning approaches, and therefore support instructional designers in the preparation of better courses. According to Litzinger and Osif, learning styles are "the different ways in which children and adults think and learn" [32] (pp. 73).

They suggest that every student develops a preferred and consistent set of behaviours or approaches to learning.

In order to better understand the learning process, they seek to study its main processes:

- Cognition — the various means of knowledge acquisition.
- Conceptualisation — information processing. For instance, some students are always looking for connections among apparently unrelated events. For others, each event triggers a multitude of fresh ideas.
- Affective — the students' decision-making styles, levels of motivation, values and emotional predilections will also influence their learning styles.

Various attempts have been made to classify in more detail the ranges of learning styles [33]. Kolb [31] is perhaps one of the more influential thinkers in this area, and his taxonomy is shown in Figure 1.2.

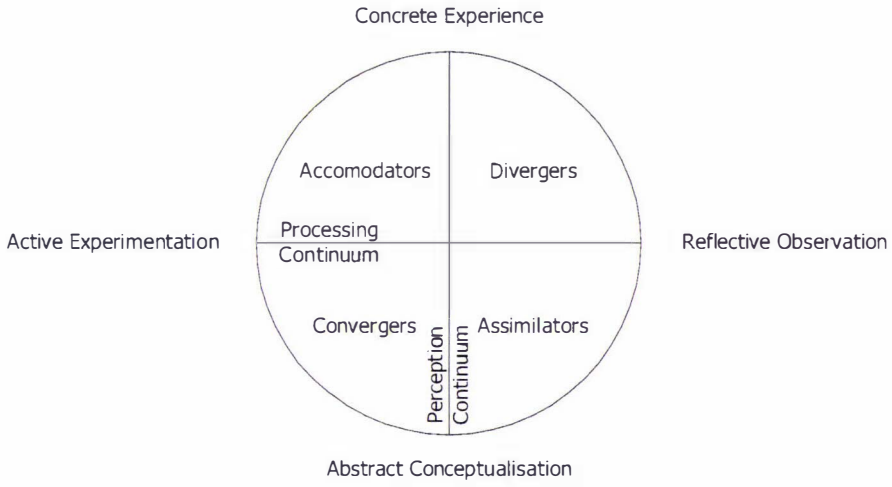


Figure 1.2 Learning styles according to Kolb [32] (pp. 79)

Kolb argued that learning styles could be seen to fluctuate on the perception and processing continuums, encompassing:

- Concrete experience — direct involvement in new experiences.
- Reflective observation — watching others or developing observations about personal experiences.
- Abstract conceptualisation — creating theories to explain observations.
- Active experimentation — using theories to solve problems or make decisions.

From this starting point, Hartman [34] gave examples of how teaching might be tailored for each of these learning styles:

- For the concrete experiencer — provide laboratories, fieldwork, observations or thought-provoking media.
- For the reflective observer — use logs, journals or brainstorming.
- For the abstract conceptualiser — give lectures, papers and analogies.
- For the active experimenter — offer simulations, case studies and homework.

In Kolb's [31] view, a student's learning style may flow within this continuum over time. However, it has been observed that students generally find affinity with and then tend to rely primarily on one style more than the others. Based on this insight, instructors should constantly aim to recognise correctly the predominant learning style of each student and tailor their delivery and instructional materials accordingly. This is where electronic learning environments provide the most promise and challenge, due to their flexibility.

Besides more discerning considerations of teaching and learning styles, practitioners have also reached a more realistic appreciation of what intelligent tutoring systems can and cannot do, and it has become apparent that the effective administration of students, authors, lecturers, courses and so on, is much more important than previously thought.

However, the effective authoring, management and distribution of high-quality educational resources cannot be overlooked. Once online lecture material is created, it must be stored and efficiently distributed to students, who must be able to enrol and follow online courses, including their associated examinations. Learning Management Systems provide the framework for creating such web environments for online instruction, including facilities such as student registration and authentication, content creation, management and reuse, quizzes, exams, report uploads, homework grading, instructor grade books, class calendars and online help [62].

Although there is general agreement that powerful authoring and learning management systems are essential, substantial courseware libraries cannot be built unless systematic re-use of modules based on meta-data and presentation standards become widely supported. Therefore, there is a growing realisation that standardisation is a critical issue to be addressed.

According to the IEEE Draft Standard for Learning Object Metadata [47], a learning object is defined as any entity — digital or non-digital — that may be used for learning, education or training. Boyle delineates an interesting coherent framework for authoring learning objects that may be deployed with different purposes and offers a number of design principles for authoring such dynamic, reusable learning objects [46].

According to Smith and Ragan, *instructional design* refers to "the systematic process of translating principles of learning and instruction into plans for instructional materials and activities" [49].

A brief analysis of representative instructional design theories, as described by Reigeluth [52], reveals that many of the current instructional design models suggest that the most effective learning environments are those that are problem-based and involve the student in four distinct phases of learning:

- Activation of prior experience.
- Demonstration of skills.
- Application of skills.
- Integration of the prior and newly acquired skills into real-world activities.

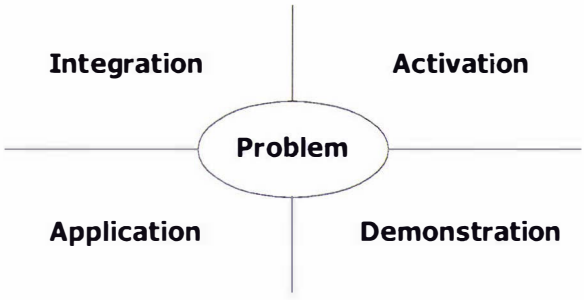


Figure 1.3 First Principles of Instruction

Consequently, to guide instructional design at a high level, Merrill [51] proposes five first principles of instruction, as shown in Figure 1.3. He suggests that learning is best facilitated when:

- Learners are engaged in solving real-world *problems*.
- Existing knowledge is *activated* as a foundation for new knowledge[2].
- New knowledge is *demonstrated* to the learner.
- The learner *applies* new knowledge.
- New knowledge is *integrated* into the learner's world.

West, Farmer and Wolff suggest that an *instructional designer*'s task is to plan the instruction so that the student can use various cognitive strategies to learn actively [50].

Liu et al. [59] have undertaken a study to learn from the practitioners the roles and responsibilities of an instructional designer in developing new media enhanced instructional materials, with particular focus on the challenges designers are facing and how they handle

[2] Merrill used the word *knowledge* in its broadest sense to include both knowledge and skill, and to represent the knowledge and skill to be taught as well as the knowledge and skill acquired by the student.

these challenges. They have found that the biggest challenges confronting designers arise in dealing with clients, balancing multiple roles, and adapting to rapid technological changes.

Dobrovolny, Spannaus, Sims and Lamos [58] also argue that many instructional design projects struggle because instructional designers who have little project management experience are required to manage them. They show that while experienced instructional designers frequently perform the role of project managers, instructional design graduate programs rarely require project management courses. However, they recommend that instructional designers should familiarise themselves with project management disciplines, as they will probably be required to perform many project management responsibilities in order to successfully orchestrate the creation of complex web-based *instructional applications*.

Buendia, Díaz and Benlloch define an *instructional application* as a set of resources and activities that implement interacting, interrelated, structured experiences that are designed to achieve specific educational objectives [56], and recommend the instructional application design architecture shown in Figure 1.4.

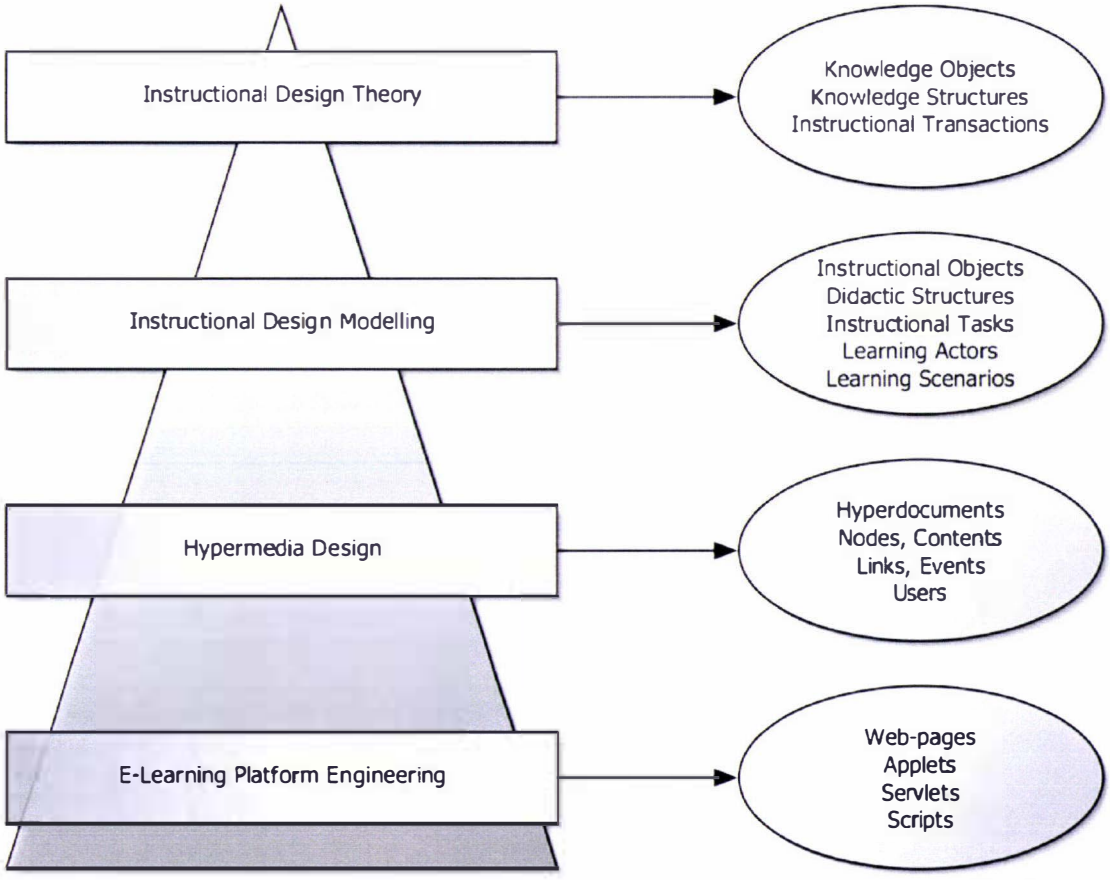


Figure 1.4 Instructional Application design architecture

Nunes et al. [54] suggest that rapid prototyping — in short, successive iterations of design, implementation, and assessment — is an ideal approach for instructional application design, as

it facilitates the integration of the different agents in educational software development: the subject matter experts, the instructional designers and the software developers.

They indicate that after each evaluation in the iterative instructional design process, it is most likely that both the modules and the curriculum will have to be remodelled regularly in order to accommodate the changing characteristics of both students and the environment.

Lecturers now realise that computer networks have opened the way for entirely new approaches involving much more communication, collaboration and teamwork. For instance, some ideas of intelligent tutoring systems have been successfully replaced by techniques involving networks. Maurer also stresses the importance of feedback, active documents and knowledge management in this context [26].

In general, the most dramatic step forward has been the recognition that individual multimedia packages cannot be as effective as they should be in the absence of powerful, networked learning environments. Although such environments are starting to appear on the horizon, given that the role and success of educational multimedia has different characteristics in schools and universities when compared to corporate applications, they are likely to evolve in slightly different patterns before perhaps converging in the longer term on a unified, standardised collaborative learning environment.

1.3 The AudioGraph Idea

The main objective of the AudioGraph concept is to capture the typical classroom lecturing experience in a form that is easy to present and deliver over the Internet. Lecturers should be able to prepare material without complex programming requirements. To this end, we should provide lecturers with a familiar environment for presenting slides, annotating them with graphical elements, and providing explanations in the form of voice annotations.

1.3.1 Vision

As discussed by Jesshope and Shafarenko, one of the crucial factors limiting the introduction of multimedia material for web-based education has been its production cost [1]. Therefore, reducing this cost may contribute to the wider, faster adoption and higher effectiveness of web-based education.

Jesshope and Shafarenko have started with the assumption that the traditional lecturing techniques used by most professors in virtually all the traditional universities, which involve using voice comments to discuss slides and whiteboard diagrams and notes, should be used as a basis for simple, low-cost multimedia lecture recording [1]. After due evaluation, if this approach were to prove insufficient for effective web-based lecturing, further enhancement with

more sophisticated — and considerably more expensive — forms of information exchange such as modern collaboration tools, video or advanced animation should be pursued.

In discussing the difficulties related to software technology, Brooks distinguished between *essence* — the difficulties inherent in the nature of software — and *accidents* — those difficulties that today attend its production but that are not inherent [132]. He also observed that the hard part of building software lies in the specification, design and testing of conceptual constructs rather than in the labour of representing it and testing the fidelity of the representation [133].

By developing the AudioGraph idea, we have looked upon the difficulties related to the creation of computer-based educational material — as it could be regarded as a form of software — in a similar light and sought through our work to alleviate the *accidental* complexity of media creation in order to free educators to focus on the *essential* complexity of education. Rather than asking educators to master what could essentially be seen as yet another programming environment, we hoped to be able to offer a familiar and simple framework, with a very shallow learning curve. In this context, ease of use throughout the system became one of the primary goals of our work. In addition, good quality voice, images and annotations and low bandwidth requirements for streaming the system's educational content output were sought in order to appeal to and reach the widest possible audience.

We envisaged that this approach might find applicability in some of the following areas [6]:

- Formal university lectures.
- Software or systems documentation.
- Tutorials and various other training presentations.
- Guides, such as multimedia museum guides.
- Advertisements.
- Educational animations.

In the usual scenario for traditional classroom-based education, human lecturers present their course by working their way through a presentation. The presentation may take the form of a set of PowerPoint [64] slides, be it projected in electronic form or simply using transparencies. Typically, lecturers annotate such transparencies or use a whiteboard as they are presenting the course. On certain occasions, the lecturer may even be using a scrolling transparency roll when presenting longer arguments.

The customary sequence of events is quite predictable:

- The lecturer will select a slide, and present it to the class.
- The lecturer will comment on the slide, perhaps making annotations on it.

Then, the above cycle repeats until the class finishes.

The main benefit of this approach is that the lecturer may explain or present the new concepts being taught in a multitude of ways, until the students grasp the intended meaning. The main limitation of the same approach is that the students and the teacher must be in close proximity.

The teacher's time is a very valuable resource, and she can only reach a limited number of students in each class. Therefore, finding a means of reaching more students in the same time with similar educational results is a worthwhile goal, given the importance of ongoing education in our ever more rapidly changing society.

One step towards this goal could be to record the teacher's performance and then present it to students. Students can replay the course material at their leisure, as many times as they wish, being free to focus on the issues they personally find complex and skip over easier material.

From a lecturer's perspective, what remains to be decided is the recording medium. This choice is of utmost importance; from it derive a large number of consequences, such as the cost of recording, the cost and ease of distribution, the cost of playback and the ease of use of the recorded material.

For instance, given the ubiquitous nature of VHS video players, there are already many courses recorded on videotape. However, the cost of recording the tapes and multiplying them is relatively high, when we take into account the resources required for these tasks.

For the AudioGraph system, in order to minimise costs and maximise distribution capabilities, the Internet seemed an excellent candidate for the delivery medium of choice, since it provides highly standardised and relatively inexpensive communication mechanisms. However, Jesshope and Shafarenko suggested that distribution could be accomplished via various storage media (e.g. CD-ROM) as well, or even in a mixed format, using the web for the most up-to-date material and storage media for rapid access to infrequently changing reference content [1].

The most widespread means of information distribution over the Internet involve using the HTTP [184] protocol for distribution and web browser applications for viewing electronic documents, typically authored or presented using the HTML [190] language. As a result, it would be most convenient if we could prepare annotated lecture presentations into a format that would be readily delivered using the above standards. For this purpose, we would require a recording application, to capture and structure annotations into coherent presentations.

Since the HTML language specification [191] provides the ability to employ *user agents*, it is possible to develop web-browser plug-in components that can render arbitrary data streams, and therefore we could create viewing applications that can be integrated seamlessly into most web-browsers currently available.

1.3.2 Intended Users

Dabbagh suggests two main classes of lecturers are making use of technology to enhance education [35]:

- *Information technologists*, which think like engineers, and work to enhance information processing systems through technology applications. They typically take a systems approach by focusing on transaction-driven, query-based information retrieval, and by consistently seeking accuracy and efficiency in delivering, organizing and managing information.
- *Instructional technologists*, which think like educators and apply technology to enhance learning systems, as cognitive tools. They typically take a pedagogical approach by focusing on conveying content-driven information, with emphasis on the effectiveness of achieving specific learning outcomes and the instructional impact of delivering, organizing and managing information.

In the past, instructional technologists were at a disadvantage, as the application of technology required to serve their aims required either considerable programming expertise or the help of information technologists. In an attempt to address this imbalance, our efforts primarily aimed to assist the instructional technologists, by providing them with an easy-to-use framework that could be used in the process of designing, constructing and experimenting with effective learning environments.

1.3.3 Starting Context

The term "audiograph" and the basic concept behind it are not original to the author. Although Professors Jesshope and Shafarenko proposed the original "audiograph" idea and described a prototype system developed at Surrey University [1], the actual term *AudioGraph* was not used in that paper. Instead, it first appeared in a formal evaluation study of the prototype, conducted by Segal [2], and its spelling at the time was *Audiograph*.

The author of this Thesis joined the Surrey University research team at a very early stage and was involved in the assessment of the prototype. The first references to the *AudioGraph* term in its current spelling have been related to early trials of our work on version 1.0 of the system, as published in [7] and [8], after Professor Jesshope and the author had joined Massey University. Throughout the remainder of this paper, we will use the terms *prototype* or *AudioGraph prototype* to refer to the system described in [1], while the term *AudioGraph* itself is consistently related to our research efforts.

Figure 1.5 shows the Visual Basic recorder prototype and a sample presentation in a web-browser window, which used a Java applet [209].

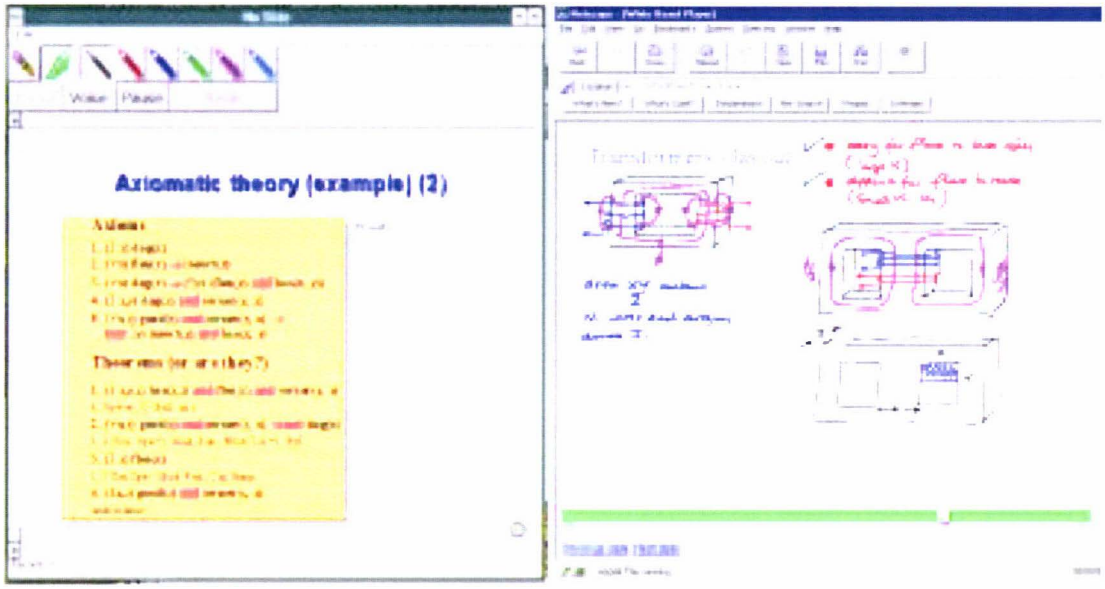


Figure 1.5 The AudioGraph prototype [1] recorder and plug-in

The original AudioGraph prototype system provided not much more than a proof-of-concept. Its user interfaces were rudimentary, the visual annotation capabilities were very limited, and the sound quality was poor. In short, although it put into practice a useful concept, it was not satisfactory, and its shortcomings motivated our research. We will show how we have built upon the original ideas, extended the main concepts and substantially enhanced the capabilities of the system.

In the following sections, we will critically examine the educational, conceptual and technical aspects of the prototype and then analyse their implications for our research.

1.3.3.1 Educational Aspects

It has been argued by Draper [24] and Gunn [25] that technology-enabled educational systems can only be evaluated realistically within the context of an actual course, using students actually registered on that course.

In this spirit, a rigorous study of the use of the AudioGraph prototype was undertaken at Surrey University by Segal [2] and it has particular significance as it involved a relatively large sample of students, making it a noteworthy assessment of this mode of lecture delivery.

For this study, the subjects were drawn from two courses in Discrete Mathematics — one for first year undergraduates and the other for M.Sc. students — presented at Surrey University by Prof. Shafarenko. Students on both courses were provided with an explanation as to how the course was going to be presented and supplied with printed notes. They could access the courseware either by using networked workstations in a laboratory, or on their own machine using a CD-ROM.

The evaluation centred on the analysis of data from questionnaires, and a response rate of 100% from the postgraduate students and about 25% from the undergraduate students was obtained. Segal designed the questionnaire using preliminary interviews with a handful of undergraduate students together with Prof. Shafarenko. The questions were both closed and open, that is, requiring answers to be chosen from a list of provided alternatives — closed — or inviting students to make their own comments — open. We will present a small extract from the evaluation study [2] here.

Since the main aim of the web-based lecturing system is to provide an alternative lecture delivery mode, the first question asked of respondents was as follows:

> *Overall, how do you feel about using the software compared with attending conventional lectures? (Please ring the appropriate answer)*

The responses chart, in percentages, is given in Figure 1.6. It can be seen that the postgraduate students, who seemed to be more self-motivated, rated the web-based lectures somewhat better than conventional lectures on average, which is quite a surprising result, especially in the light of the 100% poll response.

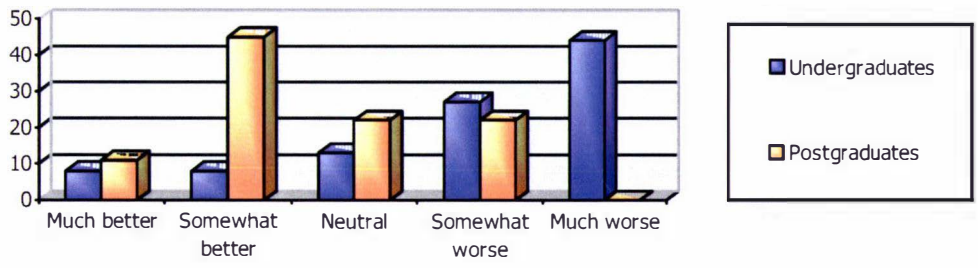


Figure 1.6. Students' perceptions (in percentages) of using the web-based lectures contrasted with attending conventional lectures (25 undergraduates and 9 postgraduates)

Significant numbers of undergraduate students also found the mode of presentation better but on average they preferred the conventional mode of delivery.

The evaluation has proven very valuable since it has offered vital clues for the enhancement of the AudioGraph concept, and led to the design of several possible alternative approaches that should increase the level of satisfaction with the AudioGraphic presentations.

The students were also asked to make open comments on the disadvantages and advantages of the mode of delivery and some of these responses are given below.

Disadvantages:

- [Need a] schedule to show how far in front/behind you are. (Undergraduates)
- As it stands, the course puts very little pressure on students to study [regularly]. (Undergraduates)

- There is no incentive to "keep up". Exam results will probably suffer from people rushing study at the end of semester. (Undergraduates)
- [Should be] scheduled into weekly amounts to be completed with tests after each weekly section so we can tell how we are doing, but these marks don't count (Undergraduates)
- This has the disadvantage in that it may be difficult to keep in pace with the course. (Postgraduates)
- The lecturer should monitor the progress of the student, as there is no formal lecture. (Postgraduates)

Advantages:

- Verbal explanation of obscure points. Seeing problems worked out step by step with explanations. No distraction from other people. (Undergraduates)
- It combines both the advantages of written material (i.e. being able to refer back to previously covered material at own pace) as well as allowing some form of interaction (limited!) in the form of the lecturer's voice. (Postgraduates)
- As a part-time student, this is a tremendous facility as it allows one to prepare in advance, allowing in principle at least to get the most of lectures/tutorials. (Postgraduates)
- More dynamic than a text book and therefore better able to keep ones attention. (Postgraduates)

We can observe that adherence to good instructional design principles can help alleviate virtually all of the perceived disadvantages. For instance, especially for the undergraduates, releasing AudioGraphic material in short iterations may help to control the pace of the course and encourage the discipline of regular study. Periodically verifying comprehension and the availability of the lecturer to support students are equally important.

Also, reviewing the listed advantages, we can conclude that some of the educational benefits originally hoped for — such as an enhanced lecturing experience — have been perceived by the students and therefore have been realised.

1.3.3.2 Conceptual Aspects

Perhaps the most important design decision of the prototype was the adoption of a single presentation stream, with sequenced audio and graphical annotations. No concurrency of presentation of any kind was envisaged.

For the prototype [1], the individual units of presentation were named *frames*. Each *frame* could display a single image, which was essentially its background. Emphasis was placed on the fact

that images used for backgrounds should be scalable, such as vector Windows Media Format (WMF [257]) drawings produced by Microsoft PowerPoint [64]. However, although full scalability was originally sought so that students might be able to take advantage of the small resolution monitors available at the time (1996-1997) by zooming the playback applet to fit their available display areas, this capability was not implemented by the prototype.

The recording experience focused on a single *frame* per editing session. Multiple undo and redo functionality was available.

1.3.3.3 Technical Aspects

As can be seen in Figure 1.5, users have at their disposal an erasing tool, a highlight tool and six free-hand drawing pen tools — for black, red, blue, green, magenta and cyan. The undo, voice, pause and redo buttons complete the array of available controls in the recorder.

When preparing *frames* for web-based delivery, each of them was transformed into a Java applet [209] with associated command and audio files, with an associated wrapper HTML [190] page created in direct sequence to the previously prepared *frame*. If adjustment were required in order to prepare a new lecture as a separate sequence of frames, the generated HTML files would need to be edited and adjusted accordingly.

The background images were exported in the proprietary GIF format [225]. The voice patches were exported in the Sun *au* sound file format [233], which used 8-bit samples at 8kHz, using μ-Law compression (ITU-T G.711 [234]), as supported by Java [208] at the time.

The applet in Figure 1.5 uses a green scroll bar to indicate playback progress. When the presentation was stopped, the bar would turn red. Once stopped, navigation through the *frame* material was possible by scrolling. No fine-grained — one annotation at a time — navigation controls were available. The presentation would be started and stopped by clicking anywhere on the display area.

1.3.4 Analysis

Given its characteristics, the AudioGraph prototype could be considered very simplistic, and closer observation would reveal a number of user interface shortcomings. In the following paragraphs, we discuss the conceptual and technical aspects that we have considered and present the main drawbacks we have identified as we started the work described in this paper.

Given its simplicity, and therefore likely appeal to the target user base, the single presentation stream — with sequenced audio and graphical annotations and no concurrency requirements — seemed an excellent foundation to build upon.

However, the prototype gave no feedback about the types of annotations present in the recorded stream. Annotations could not be selected, moved on screen or re-arranged in the presentation stream. We considered as a severe limitation the fact that the presentation stream could only grow like a stack, with the latest annotations being undone or re-done as necessary.

The prototype Java applet [209] could only start the presentation once all its code and associated files were downloaded to the client computer. This delay could be avoided if presentations could start playback as soon as possible, while the rest of the presentation was still downloading. This capability is called *streaming*.

The prototype recorder provided no means of authoring annotations such as straight lines, rectangles, arcs or ovals. Also, it provided no means of easily creating filled graphics, such as filled rectangles, arcs or ovals.

The range of colours in the prototype was drastically limited to just six — black, red, blue, green, magenta and cyan. A further significant drawback was that the authoring tools imbued annotations with their attributes upon creation, which then became immutable.

The prototype's choice to use single images as the backgrounds for its *frames* seemed unnecessarily limiting, as multiple images may be used to good effect for presentation and animations. Also, the sound quality left much to be desired.

1.4 Research Objectives

Our research has been driven primarily by the following objectives:

- Our primary aim has been to provide lecturers with an effective and easy to use system that would assist web-based teaching and learning, allowing them to focus on the application of relevant educational principles rather than requiring them to master arcane technical complexities.
- This primary aim was to be fulfilled by building on the conceptual and technical requirements arising from the AudioGraph idea and its prototype implementation, supported by evaluation of other existing systems and approaches.
- The resulting web-based lecturing system was to be developed by following sound software engineering principles, with specific emphasis on human computer interface design issues.

From a practical perspective, the system had to be easy to use and facilitate the creation of efficient presentations, suitable for efficient distribution over the web, without substantial degradation of the user experience even over low-bandwidth connections. We also envisaged providing a good quality speech soundtrack and very good quality visual annotation capabilities, including full colour and transparency support.

Once in use, if our system can successfully demonstrate such characteristics, we will have succeeded in creating a web-based lecturing system that will be of high value for web-based teaching and learning.

1.5 Methodology and Structure of the Thesis

In this chapter, we have reviewed the motivating forces behind our work, and indicated our specific area of interest — web-based lecturing systems. To provide a wider perspective and develop a better appreciation of the complexities of effective web-based education, we briefly introduced its main facets and discussed their interrelationships.

We then put forward the AudioGraph idea, exploring its vision and intended users. We have clarified the context of our work by introducing the precursor AudioGraph prototype and we have analysed its positive and negative aspects.

Based on this foundation, we have formulated our research objectives, which pointed out the necessity of an assessment of existing lecturing systems in order to validate our perspective and inform our design decisions.

This analysis would place us in a position to crystallise our AudioGraph concepts and requirements and proceed to the design, construction and evaluation of our system.

For the design and development of the AudioGraph system, from a software engineering perspective, we intended to use an iterative, incremental software construction methodology, similar to the recommended approach popularised by the Unified Process [146]. The iterations were to contain the following activities:

- Requirements gathering and analysis. As appropriate, user interface design sketches should be used for supplementing and clarifying requirements.
- Design, continuously focused on reducing risks and executable architecture. Rather than exhaustive design up-front, emphasis should be placed on creating sufficient design artefacts to support the delivery of executable software in short iterations — from a few days to a couple of weeks.
- Construction. In each of the iterations, either a set of new features should be developed, or the architecture of the system should be re-factored [205] in order to improve quality and performance characteristics.
- Testing. In each of the iterations, adequate tests should be run to verify software quality. Attention should be paid to supporting capabilities for regression testing.
- Evaluation. After each of the iterations, progress should be evaluated and activities for the next iteration should be estimated and prioritised.

In Chapter 2, in order to give us a wider perspective and allow us to make informed design decisions in support of our primary research objective, we will review the main authoring approaches for online lectures, focusing on a number of comparable systems with web-based lecturing applicability, assessing their strengths and shortcomings.

We introduce in Chapter 3 the concepts, capabilities and characteristics of the AudioGraph web-based lecturing system. We seek to show how they address our research objectives and our conceptual and technical requirements while mitigating the shortcomings of the prototype and the comparable lecturing systems assessed in Chapter 2.

Chapter 4 presents the software engineering context of our work, by first exploring the implications of sound human interface design principles for our research. We then review the challenges of data compression, with emphasis on the image and speech applications most relevant in our context. In conclusion, we discuss the design patterns [127] and development frameworks that have played a pivotal role in our designs.

In Chapter 5 we have captured the software architecture of the AudioGraph system, and gave usage examples for the design patterns introduced in Chapter 4.

In Chapter 6 we discuss our conclusions, critically evaluating our research. We show how the AudioGraph has already been used in the field and present the insights of a number of informal evaluations. We then review our results and contributions from conceptual and technical perspectives, and evaluate the effectiveness of our software engineering methodology.

Looking forward, Chapter 7 shows a number of promising future research avenues. We suggest a number of evaluation and iterative improvement opportunities. We recommend that effective learning management systems integration approaches and alternative distribution formats should be investigated and conclude with suggestions for extended media support and other technical improvement opportunities.

2 Existing Lecturing Systems

In this chapter, we examine the main authoring approaches for online lectures, after we briefly consider educational systems taxonomies to help focus our review and later assessment.

We then concentrate our attention on assessing a select number of systems with application in a web-based lecturing context, in order to inform the design of a system that would meet our research objectives — discussed in Section 1.4 — while emulating their strengths and mitigating their shortcomings as well as possible.

2.1 Educational Systems Taxonomies

Bruce and Levin have proposed a taxonomy [37] based on a four-part division suggested a few decades ago by Dewey [38]: Inquiry, Communication, Construction, and Expression. They have also tested its utility by using it to classify a set of "advanced applications" of educational technologies supported by the United States National Science Foundation.

In brief, this taxonomy covers:

- Media for Inquiry:
  - Theory building — technology as media for thinking.
  - Data access — connecting to the world of texts, video, data.
  - Data collection — using technology to extend the senses.
  - Data analysis.
- Media for Communication:
  - Document preparation.
  - Communication — with other students, teachers, experts in various fields, and people around the world.
  - Collaborative Media.
  - Teaching Media.
- Media for Construction:
  - Control systems — using technology to affect the physical world.
  - Robotics.
  - Computer-aided design.
- Media for Expression:
  - Drawing and painting programs.
  - Music composition, creation, editing and accompaniment.
  - Interactive video and hypermedia.
  - Animation software.
  - Multimedia composition.

Another taxonomy [39], put forward by the Educational Technology Research and Assessment Cooperative at Rice University [40], divides the field as follows:

- Intelligent systems.
- Computer-supported collaborative learning.
- Technology-supported learning environments.
- Visualizations & simulations.
- Cognitive tools.
- Hypermedia / Multimedia.
- Computer-Aided Assessment.
- Authoring tools.
- Architectures for learning systems.
- Digital repositories.
- Learning devices / learning appliances.
- Learning interfaces.
- Educational facilities.
- Learning technology systems.
- Networking infrastructure.

Although these taxonomies offer rich classification frameworks and interesting perspectives of the domain, they are too complex for our purpose of examining the context of comparable systems with applicability in the area of web-based lecturing.

Instead, we will use for this task a simpler classification, introduced by Dabbagh [35], and select a few representative solutions with which it shares some areas of commonality:

- Multimedia Authoring Systems — Multimedia-based authoring, web-enabled 'play' capabilities.
- Web-Based Authoring Systems — Web-based authoring and publishing capabilities.
- Web-Based Course Management Systems — Web integrated applications.

Some of the systems reviewed could be classified under multiple categories in this simple taxonomy. For instance, many of the web-based authoring systems have evolved predictably over time to include web-based course management components.

2.2 Authoring Approaches for Online Lectures

Traditional correspondence schools send printed course materials to students by mail. In this scenario, students learn using these educational materials and send test papers by return mail. Although this process was slow and costly, it was the best alternative available at the time that

could address the education needs of students located in remote regions, and for that reason it is still used in many locations where electronic communications are either not available or not cheap enough yet.

With the advent of the Internet, communication has become significantly cheaper for very large segments of the population in many countries, and possible on a scale considerably larger than at any time earlier in history. Electronic communication lends itself to different approaches for supporting education and distributing information depending on the format of the material and the degree of interactivity sought.

2.2.1 Multimedia Content Authoring

Developing educational content rich in interactive multimedia elements by using plain Hyper Text Markup Language (HTML [190]) and leveraging the scripting abilities provided by web-browsers is relatively difficult and requires a substantial amount of programming effort, when done without resorting to specialised tools. Given that one of the major requirements of web-based education is the ability to develop good quality course material quickly, it conflicts with the difficulty of authoring such HTML-only [190] courses as soon as more interactive features are desired.

As a result, various solutions have emerged to address the multimedia limitations of plain HTML [190] content delivery. Fortunately, HTML [190] allows the use of *user agents* [191] and as a result, web-browsers provide the use of plug-ins, which are software components that know how to render custom data streams in a form meaningful to the human user. With such capabilities available, it becomes possible to create special applications designed for the efficient creation of multimedia documents, and have their output distributed over the web to clients that simply require the suitable browser plug-in in order to view such material.

Perhaps the most popular rich multimedia file formats for web delivery are Flash [86] and Shockwave [91] from Macromedia.

Using Flash, authors may create lightweight web applications such as animations, complex diagrams, or even graphical user interfaces for immediate deployment over the web. The Flash player plug-in [87], available for most popular computer platforms such as Windows, Macintosh, Pocket PC, OS/2, Sun Solaris, Linux and SGI IRIX, is required to render Flash content. The Flash file format (SWF) [88] is freely available, as well as an associated software development kit for generating SWF files. Using this facility, other software developers may include support for Flash in their applications.

Shockwave [91] can deliver complex multimedia content, such as advanced three-dimensional scenes and games, including video content. It can support richly interactive educational

applications deployable across multiple mediums, such as the Web, CD, DVD or even kiosk-type applications. Shockwave also supports the development of multi-user, collaborative applications.

Although the Macromedia Director Shockwave Studio [91] is the premier authoring tool for creating Shockwave content, specialised content creation frameworks such as the Authorware [84] educational content building system may have their output *shocked* — transformed into Shockwave format — for seamless web delivery. For playback, Shockwave requires either a browser-based plug-in player [92] or a standalone player, also widely available.

Virtually all the educational solutions that leverage plain HTML [190] content for the delivery of their course material can support such advanced multimedia capabilities by embedding specific Flash or Shockwave content files in their courseware. Essentially, the custom multimedia component does not change the web-based distribution paradigm, but enriches it and generally requires more effort and skill for its successful development.

2.2.2 Web-based Course Authoring

The evolution of HTML [190] has revolutionised the way information can be presented and accessed. Whereas in the past information was traditionally communicated through books and other publications, containing chapters and paragraphs, websites have been constructed from HTML [190] pages. The key to the utility of the web is the use of hyperlinks and the Uniform Resource Identifiers [196].

Numerous educational systems — such as WebCT [68], TopClass [70], Pathlore [108] or Ucompass Educator [119] — seek to utilise the popularity of HTML [190] and focus primarily on migrating existing documents into educational websites, providing various means of organizing the material into courses. Generally, a portal-style interface is used to present course material in a structured fashion. Many systems — like ToolBook [96], Pathlore [108] or QuestionMark [114] — also provide means of testing students and tracking their performance.

The VCampus Corporation [111] chose an interesting variation on the educational website theme for its course delivery approach, by using the *building* metaphor. The VCampus solutions are structured around virtual *buildings*, where different interactions are possible and specific sets of information are stored. This approach mimics the environment found in most universities, providing students with a familiar set of concepts that would help them find their way around the educational website.

For example, the *Registrar Building* allows students to sign up for available courses. In this building, administrators can add and update courses and admit students into their registered courses. The *Classrooms Building* lets students follow their registered courses or check their

grades for any of their currently registered courses. The *Faculty Building* facilitates interaction between students and instructors through various messaging capabilities. Faculty office-hour schedules are also available in this *building*. The *Commons Building* provides students with means of interaction with other students, and in addition offers access to other diversions as well. The *Information Building* can be used by administrators to post news, announcements, and frequently asked question lists to keep students informed about their training options.

Virtual U [104] from Virtual Learning Environments [103] uses a similar approach, providing a customisable virtual campus for online, distributed learning. It is intended to allow program developers to focus on the learning models and instructional design of their courses, instead of the mechanics of program delivery and management. The emphasis in Virtual U is placed on *conferences*. Students and instructors interact through *conferences* that can be accessed through a web interface, without requiring any specialised software beyond the web browser.

Using only the facilities provided by HTML [190] itself leads to a limited amount of interaction with the student. HTML [190] has evolved from an initial focus on content presentation towards a more disciplined approach for specifying document structure. Current standardization and modularisation efforts [195] surrounding XHTML, such as XHTML 1.0 [192], XHTML 1.1 [193] and Cascading Style Sheets [194] strive to divorce the content presentation from the document structure, and advance toward modularisation in order to provide a framework for defining families of markup languages derived from HTML [190]. By allowing different platforms to support different markup language modules as required, this approach aims to support the standardised delivery of content to a variety of devices with wildly different capabilities and requirements.

Markup languages still have certain limitations, since they were originally designed to simply present data without specifying interactive features. Although HTML [190] and web-browser scripting add a degree of interactivity to the browsing experience, taking full advantage of these facilities generally requires a considerable amount of programming. In order to provide richer multimedia material, it is possible to resort to using specialised *user agents* — web browser plug-ins — that know how to interpret custom data streams. Therefore, it is not reasonable to expect lecturers to spend significant amounts of time delving into web programming details in order to develop engaging educational content for students.

2.2.3 Customised, Interactive Course Authoring

Other solutions, perhaps seeking a more interactive or customised experience, take an even more radical approach to educational content delivery, sometimes developing entire custom infrastructures for course deployment. Such approaches typically require custom software

applications for viewing or participating in courses, and in many instances, specific hardware such as digital video cameras, microphones and speakers must be used as well.

For instance, Course Reader [105] from Virtual-U [104] is a custom application that students and instructors must install on their computers. It could be seen as a specialised form of e-mail client, as it is intended to be used primarily in regions that have expensive Internet connections, undependable electrical supplies or high communications costs.

Students or seminar participants connect to the Internet, typically through a telephone line, which in some regions might prove relatively expensive. They start the Course Reader and quickly download messages waiting for them in the educational seminars. Once the messages are downloaded, the Internet connection may be interrupted.

Even while offline, students can then examine the latest messages they have just received, and compose replies at their leisure. Once all the materials are ready, students may reconnect to the Internet, use the Course Reader to send their new messages into the seminars, and then they may disconnect again. Using this asynchronous messaging approach, students can participate in courses or seminars with minimal use of the Internet connection.

The Course Reader makes use of an Internet-standard messaging system called a newsgroup, and it can work with any newsgroup system in the world. Virtually all universities and many other organizations have newsgroup servers. If an organization does not have access to a newsgroup, it can install a newsgroup server program if it has a computer with a constant connection to the Internet.

Each course is organised as a separate newsgroup. Courses can then be subdivided into sections called seminars — as newsgroup threads. The Course Reader facilitates communication among students and instructors by allowing them to distribute their messages as required among the various seminars.

For organizations that can afford access to high-speed networks, their focus would naturally shift toward much more interactive solutions. For instance, the Interwise Enterprise Communications Platform [120] delivers five basic modes of communication that facilitate live interaction, collaboration and learning across an entire organization.

Each communication mode, characterised by the number of participants, the nature and depth of interaction is served by a specialised application that provides the adequate toolset and interface.

The Interwise Communications Centre (ICC) is the web-based gateway that makes access to any of the five event modes possible, providing students with a personalised view of all Interwise events in one central location.

The iMentoring tool and delivery mode is ideal for live one-on-one or small group tutoring and counselling sessions, either previously scheduled or set on-demand. The communication tool provides a display area where presentations or documents may be shared among the participants. The mentor may also introduce graphical annotations in the display area. This delivery mode may also use a two-way video and audio link.

The iMeeting tool and delivery mode is most effective for ad-hoc one-on-one or small group meetings that may also be initiated and accessed directly from Microsoft Outlook. This delivery mode offers similar features to iMentoring, with the added capability that multiple video and audio links are supported.

The iClass tool and delivery mode supports the classic mode for formal training. These moderator-led eLearning events are designed to be live and highly interactive in the quest for facilitating skills and knowledge transfer as easily as possible. This delivery mode offers similar features to iMeeting, and is intended for use by larger groups of students.

The iSeminar tool and delivery mode is designed to serve hundreds of participants. This mode enables seminar-style delivery and online interaction. It is an excellent medium for disseminating media-rich content to large groups when lower levels of interaction are required.

The iCast tool and delivery mode is an online broadcast mode that enables consistent, simultaneous communications and information sharing with thousands of participants where little or no participant response is required.

Web-based learning solutions that focus on vertical markets have emerged as well. For instance, Eloquent [115] offers custom applications that provide a comprehensive, closed-loop solution for posting, accessing, and measuring use and understanding product and sales information. Their solutions are targeted specifically at large organizations that need to orchestrate product launches across widespread locations and must train a large sales force.

Other organizations, such as FirstClass [100], SiteScape [106], ChatSpace [116], Convene [117] or Placeware [118] have focused on communications software solutions, based on the live interaction paradigm. Although the emphasis is placed on communication, these systems may be used for educational purposes. For instance, the Collaborative Groupware and Unified Communications from FirstClass [100] solutions are combined to provide users with a Common Communications Platform. The Collaborative Groupware technology is the foundation of the FirstClass Platform [100] that enables users to share information rapidly through email, conferences, directories, chats and scheduling capabilities.

When the FirstClass Unified Communications [100] solution is integrated as well into the overall system, users are able to access information and messages — voice, email and fax —

from their unified mailbox via the device of their choice including cell-phone, telephone, computer and e-mail, from virtually any location.

The Pathlore Virtual Classroom [109] and the LearnLinc Virtual Classroom [101] are excellent examples of custom solutions focused on live communication that are also geared primarily towards education, and in the case of the Pathlore solution may be integrated with the company's Learning Management System [108] offering as well. The LearnLinc Virtual Classroom is a complex solution that offers a multitude of interactive learning tools, such as "hand-raising", "floor control", "shared pointer", "assistant instructor", "record and playback", "breakout groups", "question and answers", "feedback", "text chat", "picture ID" and "glimpse" [102]. In the following paragraphs, we will briefly touch on the details of these tools, illustrating their purpose. Any web-based education solution seeking to use a highly interactive communication approach is likely to provide similar capabilities.

LearnLinc [101] instructors have the ability to monitor the list of students participating in their LearnLinc [101] class. Students may click the *hand raise* icon, and as result, the instructor can see a hand icon appear next to the student's name in the list. In such instances, the instructor can delegate control of the LearnLinc classroom — including course content and audio conferencing — to that student, who can now address the entire class.

LearnLinc [101] classes currently accept two classroom control and coordination — *floor control* — policies: instructor-led and open discussion. The specific *floor control* policy prescribes three factors during LearnLinc classes:

- Who may launch applications for the whole class
- Who controls the course content area
- Whose voice is broadcast to the rest of the class

Instructor-led *floor control* maximises the control of the instructor. The instructor may decide who has the floor, when to give the floor to a student, and when to regain control of the floor. The instructor may also use a *privacy* feature when she wishes to navigate through the course content or use any of the annotation tools without being automatically synchronised with the students. Students may raise and lower their hands to be recognised by the instructor when seeking to address the entire class, replicating the traditional classroom experience.

The open-discussion *floor control* policy has similar features to a roundtable discussion. In such an open discussion, all participants have the same level of authority, with no special capabilities for the instructors. This floor control policy is best suited for small groups of students engaged in highly collaborative work.

The *shared pointer* cursor may point anywhere on the screen and is continuously visible, regardless of any other interaction. This facility allows the person addressing the class to point out particular sections of the content or tool palette.

In order to assist the main instructor in a class, a number of *assistant instructors* may join the class in order to help the instructor manage interactivity with more than one student at a time if necessary. *Assistant instructors* may monitor the text chat, hand raising events and feedback notes, freeing the main instructor to focus on delivering the class.

As required, an *assistant instructor* may alert the instructor to interesting student responses or reply using private text chat to individual students while the main class is running its course.

Using the *assistant instructor* feature effectively provides a hierarchical interactivity infrastructure that may cater for different learning styles and student capabilities. Interaction with students is maximised, increasing the effectiveness of instruction for individual students without distracting the rest of the class unnecessarily.

The *record and playback* capability refers to the fact that live LearnLinc classes may be recorded at any time for later playback. Instructors or students may record a live class, and use it for offline instructor-led training or self-paced study. All the audio and visual interactions are recorded, such as hand raises, feedback, text chat, and synchronised courseware. The recorded material can be streamed over a network connection offering at least 26kbps bandwidth, or it can be replayed locally.

LearnLinc [101] instructors may set up *breakout groups* to facilitate smaller team discussions, role-playing, and brainstorming. The students in each group may be specifically designated or chosen at random. Such *breakout groups* can even be set up on the fly, lasting for a predetermined duration, after which the students are returned automatically to the instructor-led class.

The *question and answers* capability allows instructors or even students to present multiple-choice questions to the class. The person addressing the class may ask one question at a time with up to five multiple-choice answers. Questions may be prepared beforehand or composed spontaneously during class. The answers are aggregated in a graph displaying the distribution among the various choices.

Instructors can use the *feedback* facility [102] to maintain constantly an appreciation of the pace of the class. Students have a chance to affect the delivery of the class by giving feedback answers such as *faster*, *perfect*, *slower* or *please review*. Similar to the *question and answers* capability [102], the instructor can monitor the percentages for each type of response and adapt his course delivery accordingly. The same mechanism may be used for other types of feedback, such as agreement or assignment progress status, and can be extended as required.

The *text chat* is a communication tool [102] that allows class participants to exchange ideas and comments in text form, without breaking the flow of the class. Private messages may also be exchanged between the students and the instructor or assistants, to allow students to express questions they may not have felt comfortable sharing in the public forum. All participants may cut, copy and paste text to and from other documents into and out of the *text chat*, and a full transcript of the text generated during class may be saved or printed.

The class interaction can be further personalised with the LearnLinc *picture ID* feature [102]. All the class participants — instructors, assistant instructors and students — may provide small "passport photo" pictures of themselves, facilitating the development of a sense of community.

Instructors may also use the LearnLinc *glimpse* facility [102] to grab a screen capture of any student workstation. The instructor may thus examine assignments or assist students as they encounter difficulties.

2.3 Evaluation Criteria

Behaviourist, cognitivist and constructivist teaching approaches can all benefit from electronic course delivery systems to varying degrees. Although specific authoring tool features may have a certain influence, as suggested by Dabbagh [35], over the learning environments under construction, the way such systems are used in each approach is more a function of the instructional design, the content and structure of the course material and its intended delivery context rather than the specific capabilities of the solution used to prepare the material.

Therefore, while examining a select number of systems with comparable capabilities for web-based lecturing support, we will focus our evaluation on the following aspects:

- Graphical annotation approach and capabilities (including animations and images);
- Sound annotation approach and capabilities;
- Web integration approach and capabilities;
- Hyperlinking[1] approach and capabilities;
- Other distinguishing capabilities (such as scripting support, for instance).

At the end of each of the following sections that cover the categories of our chosen taxonomy, and in which we review alternative systems, we will summarise our findings for each of the above aspects.

[1] The *hyperlinking* approach is listed separately from *web integration*, as some solutions use custom, non-HTML [190] mechanisms for hyperlinking purposes.

2.4 Multimedia Authoring Systems

In this section, we will examine the Microsoft PowerPoint [64], Impatica [66] and HyperCard [75] media authoring systems.

2.4.1 PowerPoint

Although Microsoft PowerPoint [64] is not intended as a specific educational content authoring solution, many lecturers have developed and delivered traditional courses using it.

PowerPoint [64] presentations use the book/page metaphor. In other words, the presentation file can be viewed as a book, with the individual slides as pages. PowerPoint [64] slides may contain text, graphics, and multimedia clips. The various slide components may be animated to various degrees using animation and transition effects. PowerPoint [64] presentations may be narrated, adding a sound track to the entire slide show, just like recording a live performance.

There is no built-in support for voice-activated recording. In other words, sound is recorded even during periods of silence, leading to waste of disk space and bandwidth. Adding narration to a PowerPoint [64] presentation has the significant disadvantage of being monolithic. It is quite unusual for lectu ers to be able to deliver a full presentation while completely focused, without any slight mistakes or interruptions. Therefore, this approach to sound annotation is rather unforgiving toward the author.

In an attempt to mitigate these problems, sound clips may also be recorded and inserted individually into the presentation. However, this approach is rather cumbersome and relatively hard to control.

As the features of this application have evolved from release to release, it now provides the ability to export presentations as HTML [190] for immediate distribution over the web. However, this facility produces websites with very large footprints, effectively limiting the viability of this approach to intranets and users with very fast Internet connections. For example, a simple presentation containing fourteen slides with text and no special graphics requires 52 KB for storage as a PowerPoint [64] file. When saved as HTML [190], the resulting files occupy slightly over 1.5 MB, nearly thirty times larger than the native format.

Perhaps to address this deficiency, Microsoft has recently released the Producer [65] add-in for PowerPoint [64] to facilitate the capture, synchronization, and publishing of audio, video, slides, and images into presentations that can be delivered on-demand through a web-browser. However, the targeted audiences for this application are business users and enterprise media professionals rather than educators. Interestingly, this solution is remarkably similar to the delivery approach taken by Impatica [66].

2.4.2 Impatica

Capitalizing on the weakness that PowerPoint [64] has in exporting economical web-ready content, Impatica [66] has developed a number of solutions for efficiently packaging narrated presentations in a form that can then be played in any Java 1.0.2 or 1.1+ environments [208], and therefore distributed through any Java-enabled web browser, requiring no plug-ins.

Impatica has developed Impatica for PowerPoint, Impatica for Director, and Impatica OnCue [67]. The *Impatica for PowerPoint* and *Impatica for Director* applications simply transform the Microsoft PowerPoint [64] and Macromedia Director [91] files into a special Java [208] archive file that is read, interpreted and displayed by a custom applet [209]. In this manner, the plug-in-free operation is accomplished.

For sound annotations, Impatica relies on the presentation source being either narrated or using embedded audio clips rather than having the ability to add any narration or graphics as a post-processing phase. Given its stated plug-in-free policy, the sound quality in typical Impatica presentations is relatively poor being limited to the early basic Java [208] sound capabilities. As a result, in order to achieve acceptable bandwidth requirements, the sampling rates and sample sizes must be as small as possible, leading to poor quality sound. The early AudioGraph prototype [1] suffered from this same limitation.

Impatica OnCue [67] is an authoring solution similar to the Microsoft Producer [65] add-in mentioned above. It facilitates the production of synchronised multimedia presentations, which may use an index frame, a 'talking head' video and associated highlighted transcript, and the slide frame.

Impatica OnCue creates and translates projects that consist of:

- An *impaticised* slide presentation (.imp) file created using Impatica for PowerPoint [64] for the OnCue [67] presentation;
- A source video (.avi) file that will be translated to produce the "talking head" and, optionally, other video content;
- An *impaticised* (.imp) file for the user interface template "customised" by Impatica Customer Support personnel to include the selection of background colour and insertion of a customer-provided logo;
- A marker (.txt) file to indicate video synchronization points; and
- A "tagged" (.txt) script file that will be automatically "timed" during the translation process in order to provide synchronization information for use during playback. The script file also provides the interactive presentation index entries and the searchable narrative transcript.

The presentation produced by OnCue is organised as a collection of component files, representing the PowerPoint [64] slides, video, user interface template and narrative script components. The Impatica OnCue presentation and any other *impaticised* content is read by "ImpVideoPpt", Impatica's Java player applet [209], and played in real time while the files stream over the Internet.

OnCue presentations can be played from any HTML [190] web pages or they can be embedded within e-mail messages — HTML-capable e-mail clients are required to make full use of this facility. No special helper application software or plug-in apart from a Java 1.0.2 or 1.1+ [208] environment is required for the playback configuration. There are no special requirements — e.g. streaming support — for the server on which the presentations are hosted either.

OnCue can be used to provide compressed video-only presentations as well. Although this is an interesting feature, the likelihood of widespread authoring and use of video-only presentations in an educational setting is quite low, since good quality video material is most likely to continue to be harder to create and edit than slide-based presentations.

2.4.3 HyperCard

The authoring paradigm used in HyperCard [75] is very similar to the one used in ToolBook [96], and it also provides a scripting language called HyperTalk [78]. In HyperCard, documents are called *stacks*, and *stacks* contain *cards*. Various objects — text, buttons, graphics, images, video clips, QTVR [231] scenes, and so on — can be placed on the cards, and scripts can be associated with each of these objects. Recent versions of HyperCard [75] are fully integrated with QuickTime [229], leading to very powerful multimedia capabilities.

Cards contain a background and a foreground layer. For better efficiency, multiple *cards* can share the same background layer, providing a means to achieve a similar look-and-feel with little effort. *Card* foregrounds contain *card*-specific objects and scripts. Scripts can be stored at the *stack*, *card* and background levels, as a means of controlling their intended scope. A script editor and a command window are available for editing and testing HyperTalk [78] scripts.

Objects can contain various scripts to be executed in response to events. Scripts are written in HyperTalk [78], a scripting language that closely resembles a simplified version of English. HyperTalk [78] scripts may access the AppleScript [77] scripting language facilities and can invoke native code commands and functions stored in XCMD and XFCN [79] code resources, making it possible to integrate complex processing tasks with HyperCard stacks.

An interesting aspect of HyperCard [75] is that *stacks* never need to be saved explicitly. During editing, all changes and updates are synchronised to disk automatically. Although this

arrangement might be considered to constrain users at first glance, the multiple undo functionality provides a forgiving editing environment, leading to a simplified experience.

HyperCard stacks can be published for the web using the LiveCard [80] complementary application. It provides drag-and-drop publishing capabilities, making HyperCard stacks available through a web browser interface. Robust HTML [190] code can be generated automatically, and it can be used within LiveCard [80] for further development of interactive features or simply for the static conversion of stacks.

From a course authoring perspective, the main benefit comes from the support for the interactive HyperCard features, which enables remote users to click on HyperCard buttons, select items from pop-up menus, enter data into text fields, and so on. Using these capabilities, interactive courses, quizzes and tests can be developed.

For added flexibility and performance, since HyperTalk [78] is the underlying scripting mechanism used, support for XCMD's, XFCN's [79] and AppleScript [77] continues to be available, allowing HyperCard [75] to perform complex tasks on the server and return the results to the remote users.

Despite its relative ease of use — being at least on a par with ToolBook [96] for instance — and great flexibility and extensibility of HyperTalk [78], HyperCard [75] has not gained a very wide acceptance in the course authoring arena, although it is used with success by various organizations for developing educational and training materials [76]. However, spectacular success stories are associated with HyperCard, such as the development of the renowned interactive multimedia game Myst [267].

2.4.4 Assessment

The media authoring systems reviewed in this section fundamentally share the same underlying content aggregation concept: the units of presentation are assembled into a container. For example, slides are gathered in a PowerPoint [64] presentation, as also used directly by Impatica [67] and cards are aggregated in a stack in HyperCard [75].

2.4.4.1 Graphical annotation approach and capabilities

Common Microsoft drawing and picture objects may be used for PowerPoint [64] graphics and images. The display of PowerPoint [64] annotations may be animated by careful orchestration of animation transition attributes associated with individual annotations. Support for annotation transparency and highlighting is very limited.

Impatica [67] uses PowerPoint [64] presentations and does not enhance them by adding any other annotations. Therefore, the discussion on the PowerPoint [64] capabilities applies.

Many of the early versions of the HyperCard [75] environment were limited to black-and-white graphical objects. Later versions have added full-colour capability, although native support for transparency and highlights is lacking.

2.4.4.2 Sound annotation approach and capabilities

PowerPoint [64] provides no support for voice-activated recording. Narration in PowerPoint [64] is monolithic and therefore wastes lots of bandwidth. Although sound clips may be recorded and inserted individually, this approach is cumbersome and very challenging to maintain and control.

For sound production, Impatica [67] assumes PowerPoint [64] presentations are narrated. Transcripts and synchronisation information may be used in order to prepare the *impaticised* presentation. Sound quality is somewhat degraded by using the limited sound capabilities of Java 1.0.2 [208].

HyperCard [76] stacks are static by their nature, unless extensive use of scripting is made to provide dynamic behaviour. Sound annotation playback therefore must be either specifically triggered by a user action or programmed into HyperTalk [78] scripts.

2.4.4.3 Web integration approach and capabilities

Although PowerPoint [64] does provide the capability to export presentations into HTML-based mini-websites, the standard functionality is very inefficient in terms of storage.

In part to address this deficiency, Microsoft has released the Producer [65] add-in for PowerPoint [64] to facilitate the capture, synchronization, and publishing of audio, video, slides, and images into presentations that can be delivered on-demand through a web-browser. Preparing content in this manner is relatively laborious.

Impatica [67] presentations are directly and only prepared as web pages, again through a relatively laborious preparation process.

HyperCard stacks [76] can be published for the web using the LiveCard [80] complementary application. It provides drag-and-drop publishing capabilities, making HyperCard stacks available through a web browser interface.

2.4.4.4 Hyperlinking approach and capabilities

Action buttons may be used to provide hyperlinks, either in or between PowerPoint [64] presentations, or towards arbitrary URL's [196].

No native hyperlinking support exists in Impatica [67].

Native HyperCard linking capabilities exist [76], supporting hyperlinks to other cards in the same stack or cards in other stacks. Links to arbitrary URL's [196] are not natively supported from stacks, although they can be performed using QuickTime [228] objects that can be embedded in cards.

2.4.4.5 Other distinguishing capabilities

Using Producer [65], synchronised audio, video, slides, and images can be shaped into composite presentations.

As is the case with Microsoft's Producer [65] solution, *impaticised* presentations are similar composite structures that display synchronised video, transcripts and graphical presentations.

The HyperTalk [78] scripting language provided by HyperCard [75] is quite remarkable in its similarities with a compact, simplified form of the English language. It adds powerful capabilities to the platform, allowing for instance dynamic content generation in cards based on user inputs or events.

2.5 Authoring Capabilities of Web-based Authoring Systems

In this section, we will briefly review the web-based lecture authoring capabilities provided by WebCT [68], TopClass [70], Blackboard [72] and Mentorware [74]. These systems might also be classified under the web-based course management category, as some of them have a Learning Management System focus and they have evolved over time to provide significant course management components.

However, their native lecture authoring capabilities fit the web-based profile and are also far less advanced than the systems we have listed in Section 2.6. Therefore, we have considered it best to review their authoring capabilities under this category.

2.5.1 WebCT

WebCT [68] is primarily a supplier of web-based environments for course management. The WebCT [68] solution places the emphasis on organizing the educational material, course administration and reporting. In terms of content management, all course materials can be accessed and organised through a web-based interface.

WebCT [68] provides a template-based website authoring component, focusing on the rapid assimilation of existing educational content. Although video and audio content can be incorporated into the courses, there is no direct support for interactivity within the WebCT [68] framework itself. However, substantial support is provided for developing on-line tests and tracking the results of student tests.

Course authors can generate educational material very rapidly by accessing the WebCT [68] authoring tools via a web browser interface [69]. This approach has the benefit that no extra software is directly required for authoring simple courses, apart from the web-browser. Using the same web-browser interface lecturers may create, edit and update courses, mark student assignments or learn. Dedicated software to provide all this capability is required only at the server level, which typically runs under the UNIX [207] operating system for WebCT [68] solutions, as they are implemented using J2EE [210] and usually Oracle [212] products.

The WebCT [68] authoring interface facilitates both individual course page design and customisation of the whole course by using various global attributes, such as the layout of links, colour schemes, tools available, etc. Although default options are always available, the author has full liberty to customise the student experience.

Most authoring elements, such as questions, answers and general feedback can be supplied as plain text. Alternatively, advantage may be taken of the full flexibility of HTML [191], including graphics, audio, or any other media supported by web browsers. Although course content pages must be expressed in HTML [191], no specialised editor is provided, so the author must either code HTML directly or make use of her favourite HTML editor — such as Macromedia Dreamweaver [285], Netscape Composer [287] or Microsoft FrontPage [277] — and copy and paste the end-result back into WebCT [68].

2.5.2 TopClass

The course authoring solution provided as part of the TopClass [70] course management solution provides a very similar — although perhaps even simpler — authoring experience to the WebCT approach, focusing on the rapid conversion of existing content and orchestration into course management structures.

Within TopClass, courses are collections of learning material, divided into one or more units [71]. A hierarchy of such units can be established, by nesting them. At the leaf level in this tree of units any number of course content pages and exercises — expressed as HTML [191] — can be found. As with WebCT, no dedicated HTML editor is provided, so information must either be prepared with a specialised editor, or coded directly into the provided text entry box. In a similar fashion, the full power of HTML [191] might be exploited and any other media supported by the browsers used by the target audience could be included in TopClass courses.

2.5.3 Blackboard

Another similar solution is provided by Blackboard [72]. As with WebCT [68] and TopClass [70], Blackboard 5.5 provides a web-based interface to course authoring, and offers two kinds of content areas: Specific Content Areas and Primary Content Areas [73].

Specific Content Areas typically contain Announcements, Staff Information, and External Links. These default labels may be customised for each system, either at a global level or at the course level. Specific Content Areas are sections of course material that only accept one kind of content — for example, only announcements can be placed in the Announcements section.

Primary Content Areas typically contain Course Information, Course Documents, Assignments, and Books. In a similar fashion, these default labels that may be customised by the system administrator or the course author. The majority of the course material — such as content items, folders, and Learning Units — is uploaded or created in the Primary Content Areas.

In Blackboard, the basic building block for course material is the *content item*. Each content item may contain any combination of the following components [73]:

- Text or HTML [191] information provided by the author.
- Attached files in any useful format.
- Embedded media — images, audio, video — as supported by the web browsers of the target audience.
- Attached packaged files (e.g. websites that have been packaged for import into Blackboard).
- Links to content stored on offline media such as CD-ROMs. Such content items require the presence of the offline media when using the course material, much like some computer games require the presence of the original CD in order to function.
- Links to assessments built within Blackboard.

Folders can be used to structure and organise content items. For example, authors may wish to create folder structures that mirror the chronological schedule of the course.

Learning Units are a special variety of folder, in that authors may specify a specific sequential order in which students must access the content items in the folder. This capability facilitates the creation of a pre-defined learning flow to guide students through lessons or exercises.

It is important to note that Blackboard 5.5 [72] is designed primarily as a course management system, rather than an authoring system. The accent is placed on the ease of delivery of any type of content through Blackboard 5.5, taking advantage of the security, authorisation, integration with other communication and collaboration tools, and potentially even with an institutional portal and other similar services in the Blackboard 5.5 environment.

As a result, course developers generally do not build the bulk of their educational content in Blackboard 5.5 itself. Educational content creators would typically use the tools most suited for authoring rich content in the various formats most appropriate for each specific application and upload such content into Blackboard 5.5 for delivery.

Frequently used tools for authoring educational content may include:

- Word processors and document creation software (Microsoft Word [275], Corel WordPerfect [278], Lotus WordPro [279], Adobe Acrobat [282], etc.).
- Presentation/slideshow applications (Microsoft PowerPoint [64], Corel Presentation [278], Lotus Freelance [280], etc.).
- Spreadsheets (Microsoft Excel [276], Corel Quattro Pro [278], Lotus 1-2-3 [281], etc.).
- Web page authoring tools (Macromedia Dreamweaver [285] and Homesite [286], Netscape Composer [287], Microsoft FrontPage [277], etc.).
- Graphic/image creation and editing tools (Adobe Photoshop [283], Corel Draw [278], Jasc Paint Shop Pro [288], etc.).
- Video and audio editing tools (Adobe Premiere [284], Apple iMovie [289] or Final Cut Pro [290], Sony SoundForge [291], etc.).
- Other multimedia and courseware-specific development tools (Macromedia Flash [86], Macromedia Authorware [84], Click2Learn ToolBook [96], etc.).

2.5.4 Mentorware

Mentorware [74] focuses on collaborative content development, by offering a web-based platform providing a workflow environment that enables collaborative content development in two stages.

In the first instance, special templates are used to acquire information from the subject matter experts in an organised fashion in order to create basic learning objects with metadata assignments. At the second level, online storyboarding allows course designers, graphics artists and developers to create learning-paths and develop page-level content with specialised web-oriented multimedia creation tools.

This approach is considerably more expensive than the solutions presented above since it generally requires considerably more time and the orchestrated contribution of developers and graphic artists in order to prepare the course content. Such a solution is primarily suitable for large corporations, with training budgets that might be able to cover the additional expense.

2.5.5 Assessment

Virtually all forms of media and communication can be adapted to suit educational purposes. Therefore, in most cases, commercial educational content authoring suites have chosen to leverage existing and developing media creation and representation standards, particularly HTML [191] and the associated scripting technologies associated with the common web browser applications. This approach has several benefits such as:

- The educational material generated with other standard tools can generally be seamlessly imported into such solutions.
- The distribution of the material can occur through standard applications, without the use of customised plug-ins or dedicated applications.

However, the richness and flexibility of more advanced systems is unavailable in such solutions based on the lowest common denominator in terms of browsing technology.

2.5.5.1 Graphical annotation approach and capabilities

The solutions in this category typically generate and use static content or import static or dynamic content created with other web-page authoring tools such as Macromedia Dreamweaver [285], Netscape Composer [287], Microsoft FrontPage [277], etc.

2.5.5.2 Sound annotation approach and capabilities

Similarly, given their primary focus on web-page use, no native sound support exits in the solutions described in this section. However, if desired, sound patches could be associated with HTML [191] pages and played through various methods, such as using the QuickTime [232] plug-in.

2.5.5.3 Web integration approach and capabilities

By their very nature, the solutions examined in this section are web-based, and therefore benefit from full web integration capabilities.

2.5.5.4 Hyperlinking approach and capabilities

These solutions can make full use of the set of capabilities that HTML [191] provides for hyperlinking.

2.5.5.5 Other distinguishing capabilities

Virtually all the systems reviewed in this section have rich capabilities in the area of course management and most provide full learning management systems.

2.6 Authoring Capabilities of Web-based Course Management Systems

In this section, we will survey the web-based lecture authoring capabilities offered by more complex systems that also have rich content and course management capabilities, such as ToolBook [96], Macromedia Authorware [85] and Quest [81].

2.6.1 ToolBook

The authoring components of the ToolBook [96] system from Click2learn [95] are designed to provide facilities for creating more interactive educational content, beyond what plain HTML [191] can offer.

ToolBook uses the book/page authoring metaphor. By defining pages and placing objects on them, the author creates book documents. After the objects are defined, the author can specify various actions that these objects could perform. Most objects and actions can be created and organised using the ToolBook menus. For more advanced functionality, the OpenScript [98] scripting language is available.

ToolBook [96] is an object-oriented programming environment, in the sense that the objects created through the ToolBook user interface are the building blocks of documents. For example, buttons, rectangles and fields are all objects. They reside on pages and backgrounds, which are also objects, and all of these together reside in a book, which is an object as well. Object properties define its appearance — colour, shape, text — and behaviour — what happens when the user clicks it, for instance.

ToolBook [96] has several features that assist in the application design process: *Book Specialists*, *layout templates* and *widgets*.

Book Specialists can be thought of as wizards, providing a variety of design options and functionality for defining basic book structures. There are three types of *Book Specialists* [97] available:

- Content Specialist — facilitates building books that contain text, graphics, video, audio, questions, scoring and logging.
- Test Specialist — randomises pages in a test book.
- Glossary Specialist — focuses on creating glossaries. Glossary books can be associated with content books.

Layout templates provide ready-made graphic designs for pages in ToolBook books, including navigation buttons, text fields and options for graphics fields and video stages.

Widgets are pre-prepared buttons, questions and responses, menus, graphic placeholders and so on. A Widget is a ready-made object that has predefined properties and abilities. Some of the most popular widgets include:

- Question Widgets;
- Response Widgets;
- Media Widgets;
- Navigation Widgets;
- Smart Field Widgets.

Similar to the macro-recording capabilities in other editors, a script recorder can be used to translate actions into OpenScript [98] statements, which can then be pasted into object scripts. A facility called Auto-Script is available, providing a library of frequently used OpenScript [98] methods.

Multiple objects can be made to share the same functional characteristics, by using shared scripts, which are stored as individual resources in the book and do not belong to any object in particular. Such shared scripts can be attached to any number of objects.

If this approach is used with adequate discipline, it can lead to better use of resources — as the same script does not have to be supplied in every single client object — and may simplify code maintenance since script changes need only be done in one place.

The OpenScript [98] scripting language provides significant extensibility capabilities in ToolBook [96]. Scripts — expressed in OpenScript [98] — are object properties that define the actions to be performed in the ToolBook [96] execution context. Any ToolBook [96] object can contain script properties. Scripts can be created using the Script Editor, and they can be tested using the Command Window [97].

OpenScript [98] can be extended with functionality implemented in native code — using languages such as C++, C or Pascal perhaps — by taking advantage of Dynamically Linked Libraries (DLL [152]). At runtime, applications link dynamically to the DLL's that contain the methods not directly supported in OpenScript.

Since most of the Windows programming API is available through DLL [152] function calls, OpenScript [98] can effectively invoke systems-programming tasks and interact with other applications. Therefore, DLL's [152] are a very powerful means of extending the capabilities of ToolBook [96], or any other authoring system that would choose to offer this capability.

ToolBook [96] also supports Dynamic Data Exchange (DDE [153]), which enables ToolBook to use capabilities from other Windows applications, by exchanging information or controlling them directly.

ToolBook [96] includes a number of distinct digital content creation utilities, such as various resource editors for menus, icons, cursors, graphics, colour palette editors and optimisers, and a sound clip editor.

In terms of dynamic content navigation, there are two basic types of hyperlinks in ToolBook: *jumps* and *pop-ups* [97]. While *jumps* bring a page that replaces the current page, *pop-ups* trigger the creation of a new window on top of the current page window.

As a means of conveniently specifying hyperlinks, arbitrary strings in text fields or record fields can be defined as *hot-words* [97]. They represent strings of characters defined as objects in their own right, with their own associated properties and scripts. Although any number of *hot-words* can appear in the same sentence, no two of them are allowed to overlap.

For developing flexible evaluation solutions, ToolBook provides many different types of *Question Widgets* [97]:

- Multiple Choice;
- Drag and Drop;
- Matching Items;
- Arranging Objects;
- Sequencing Text Items;
- Fill-in the Blanks;
- Selecting Text;
- True/False;
- Sliders.

In addition, further properties may be defined, such as:

- The correct and incorrect answers;
- The feedback to be given in response to student actions;
- Instructions on how to score responses;
- Time and retry limits.

For added security, answers may be randomised in multiple-choice questions.

The type of analysis of student answers may be varied based on the type of question. Response analysis for 'Fill-in the Blank' questions is clearly the most complex. The analysis engine in ToolBook allows for wild cards, it can ignore word order, capitalization and even provides a *fuzzy spelling* capability that tolerates certain kinds of misspelling [97]. Question widgets may be locked to allow students to present only one response. Score values and weights can be specified for each response individually.

Feedback can be supplied in several different formats:

- A text message, presented either in a pop-up window or in a designated field.
- An audio or video clip, presented either on the same page, a separate page, or perhaps even a different book altogether.
- An animated sequence or actions.

ToolBook [96] provides several means of navigation. For instance, page navigation provides a mechanism for specifying links from an object in the current page to a specific page in any book. The complexities of file systems begin to be felt here, as such links need to rely on specific folder structures rather than unique indexes in some form of educational content management database. To mitigate these effects to a certain degree, generic hyperlinks — that do not specify pages directly — may be created. For instance, *Next* and *Previous* buttons can use generic hyperlinks for navigation between pages in a book.

Another navigation mode is a based on the full-text search capability. Developing this mechanism requires two steps. First, the author must create an index of all the words of interest in the current application. Once this index is available, students can perform searches and based on the results, they can jump to the appropriate pages in the application. Searches can be done by word, phrase, keyword, partial word, or a combination of words, and can be limited to subsets of the information in the application. In addition to the index capability, a glossary can be included with the book.

For web delivery, ToolBook [96] course material may be delivered either as a package consisting of HTML [191] and Java [208] files or as native ToolBook files.

The ToolBook [96] books, pages and objects and associated OpenScript [98] scripts can be transformed into HTML [190] pages with associated Java [208] scripts, and require no plug-ins for viewing. When exporting ToolBook [96] books into HTML format [191], each page in the book is transformed into an associated HTML page, and the various objects and widgets become HTML components and Java [208] widgets, as possible and appropriate.

However, for the ultimate in flexibility and power, certain scripting facilities require the use of the native ToolBook [96] files, which can be viewed in web-browsers by using a plug-in called Neuron [97].

ToolBook [96] even provides an FTP [183] utility that could be used in a variety of ways. Files can be transferred to and from remote computers, and an OpenScript [98] interface is provided for custom applications. It is advisable that the security implications of these capabilities should be carefully weighed when designing ToolBook-based solutions.

2.6.2 Authorware

The Authorware [84] solution from Macromedia provides an object-oriented user interface, using sequences of objects to define the logical control or flow.

The two basic mechanisms for developing and managing reusable course structure and content in Authorware are *models* and *libraries* [85]. A *model* is a repository for course logic and points to corresponding educational content. A *library* stores the course content, such as images, textual data, audio patches, video clips or animations.

Authorware *models* represent predefined sequences of objects. Objects are represented in Authorware *models* by using icons. As such, Authorware uses the *iconic/flow control* paradigm as identified by Siglar [48]. *Models* may contain objects or references to objects.

By specifying the logical control flow-line using icons representing various content and action objects, the author creates Authorware documents, called *pieces* [85]. They can be constructed from various *models*, by simple cut, copy and paste operations. Once *models* are available in a *piece*, they can be further edited as required. It is important to note that when pasting a *model* into the *piece*, a copy of the *model* is made and therefore the original *model* is unaffected by any changes that may occur in the current *piece*. *Pieces* may be saved as *models* to facilitate reuse[2].

Authorware *libraries* [85] are collections of objects without flow constraints. Authorware *models* define the flow of the presentation within *pieces*, while *libraries* store content. Authorware *libraries* may host image, video, sound, interaction and calculation objects.

Content objects from *libraries* are introduced into *models* by simply dragging and dropping their associated icons. In such circumstances, the content is not actually copied into the model, rather a reference to the object itself. The icon names for reference objects are presented in italics, maintaining consistency with the way file aliases are presented in the Macintosh graphical user interface.

The use of *models* and *libraries* makes it easy to standardise presentation features. For instance, to provide a family of *pieces* with a common background and the same navigation buttons, all one has to do is to define a *model* that contains them and then paste it in each destination piece. The template *model* can be reused at any point in time.

This approach has several benefits: significant space savings can be achieved, as object content is not unnecessarily duplicated. The information required for defining the reference is generally much smaller than the content data[3].

---

[2] This may be thought of as an example of the *composite* design pattern [127] (pp. 163) at work (see Sections 4.3.3 and 5.7.1 for more details).

[3] Note that the AudioGraph uses a similar referencing mechanism for the image-manipulation annotations for similar reasons.

Multiple *pieces* can be updated automatically when objects in a *library* are maintained or upgraded. However, care must be taken when using this approach to avoid violating constraints in the various contexts in which the object is used. It is also interesting to note that a single *piece* may be used in various guises by simply replacing its *libraries*.

Authorware *pieces* [85] can be extended to use various external kinds of objects by using the Sprite Xtras [94] mechanism. Sprites are media and functional elements such as 3D objects, Macromedia Flash [88] animations, web browser windows, pop-up menus. The Sprite Xtras [94] functionality makes it possible to incorporate such sprites into an Authorware piece.

Although Authorware [84] does not offer a specific scripting language, it provides limited scripting capabilities in the form of *variables* and *system functions* [85].

Variables represent values that can change during the *piece* playback due to various events or actions. Users can define their own *variables*, and they can be used in conjunction with various *system functions*.

Authorware [84] creates and updates many variables automatically. For example, the system *variable* **`PercentCorrect`** automatically keeps track of the percentage of questions a user has answered correctly. Generally, the Authorware [84] system *variables* maintain information that helps determine user performance levels, construct performance reports, and control how the application operates.

Authorware [84] system *variables* can be thought of as attributes of the various types of objects that can be manipulated in *models* and *libraries* [85]. It is possible to refer to specific object attributes by using a special syntax, such as **`VariableName@IconTitle`**.

Authorware provides numerous *system functions* for manipulating values and controlling the operation of the application [85]. Functions exist for creating arithmetic, logical and relational expressions. Conditional and looping constructs may also be defined.

Special tools in the Authorware toolbox [85] can be used to select, create, and modify text and graphics. The line, oval, rectangle, rounded rectangle and polygon tools draw simple two-dimensional graphics using the popular mechanisms used by most graphic editors. A text tool exists to create and edit text.

It is sufficient to define a special text style that contains an interactivity attribute in order to create hypertext links in Authorware [84]. Applying this custom text style to an arbitrary text string and specifying the desired URL [196] destination transform it into an active, operational hyperlink.

Authorware [84] provides the following main interaction types [85]:

- Buttons, hot text, hot spots and hot objects. From a user's point of view, a hot spot, a hot object and hot text all work the same way, just as if they were buttons. Simply clicking the hot elements triggers a certain action. Hot spots make rectangular areas active, while hot objects make arbitrary areas react to mouse clicks. Hot text imbues text data with the properties of a button.
- Key-presses. The flow of *piece* playback can be paused until a key-press event is received.
- Text-entry fields. Authors can create text-entry interactions to create text fields where users can enter text. The actual text entered by the users is available as variables belonging to the appropriate objects.
- Dragging an object to a target. Various kinds of interesting questions or interactions can be set up by allowing a user to drag an object to a target area. For example, one can ask the user to create a map by dragging countries to their proper location, to assemble a machine by dragging its parts to fit together, to trace the flow of a process by arranging its parts in the correct sequence, or to reconstruct history by lining up events in the order in which they actually happened.

Authorware provides relatively sophisticated support for judging user responses to tests [85]. For instance, in order to judge responses entered by means of text-entry fields, Authorware uses a pattern matching mechanism. Using this approach, the word or phrase the user should type can be specified, allowing the system to take into account variations in spelling, punctuation, word order, etc. when determining correctness.

A number of options are available for matching patterns:

- Match At Least. This option can specify the minimum number of words — relative to the total number of words in the pattern specified by the author — that the user must enter for a correct response. When specifying patterns, words that are separated by the bar | symbol are considered separate groups. For example, this facility may be used to provide a number of different correct answers.
- Incremental Matching. If the pattern specified by the author contains more than one word, using the Incremental Matching option gives the user several opportunities to match all the words in the pattern.

There are also a number of special options [85] that enable the author to specify certain variations to be ignored or enforced in matching the user's response:

- Capitalization — Ignores capitalization. If the user enters the text in the *Pattern* field correctly except for capitalization, Authorware matches the response successfully.

- All Spaces — Makes all spaces typed by the user become significant. By default, Authorware compares user responses with the specified text word by word, ignoring white space. At least one space indicates the start of a new word; multiple spaces are ignored. The All Spaces option indicates that every space in the user's response is significant and suggests that the entire response should be treated as a single word or an unbroken string of characters.
- Extra Words — Matches correctly even in the presence of extra words if the specified word or words exist in the field.
- Extra Punctuation — This option ignores any extra punctuation.
- Word Order — Matches the words specified by the author regardless of the order in which the user supplies them.

Authorware provides the following *system variables* for tracking student performance:

- **`PercentCorrect`**, **`PercentWrong`**.
- **`TotalCorrect`**, **`TotalWrong`**.
- **`FirstTryCorrect`**, **`FirstTryWrong`**.
- **`JudgedInteractions`**.
- **`JudgedResponses`**.

These *system variables* can be used to provide feedback or gather scoring information.

Authorware also provides a number of branching options [85] for directing the flow of a *piece*. These may include:

- Sequential Branching. Decision structures can be set up with a number of alternative, mutually exclusive flows. The first time playback flow passes through such a structure it takes the path on the left in the visual representation used in the model. The next time it passes through the structure, it takes the second path, the third time, the third path, and so on, always sequentially following the order specified by the author from left to right. When the last flow on the right is taken, the cycle starts again with the flow furthermost on the left. Such constructs could be useful for defining alternate explanations for complex points in a course.
- Random Branching. This branching method is a variation on the sequential branching approach, without the limitation of choosing flows in a sequential fashion from left to right. When using random branching, the application will select a random flow path at each pass through the decision structure. It is possible to specify whether the next random path should be selected from all paths, regardless if it has already been visited,

or it should be selected only from the yet unexplored paths. Such structures could be useful in randomising questions in a quiz section.

- Branching based on a *variable* or expression. At times, the author may wish to branch the flow of execution based on certain *variable* values or expressions. Such structures could be used to customise the flow through a course based on the accuracy of response to test sections for instance. If a certain student has mastered the current course section to a satisfactory degree, no review is necessary, and the student can proceed directly to the next course section. If another student has failed to provide enough correct answers, she may be guided to review certain sections in the course, as required, until her answers prove satisfactory.
- Branching on the number of tries. The author may limit the number of times a certain branch can be followed. Using this mechanism, the author can restrict the number of chances a student gets at answering a particular test correctly.
- Branching on a time limit. The author may specify a certain forced branch to be taken after a predefined time-period has elapsed. For instance, it is possible to limit the amount of time a user has available for completing a test.

Authors can include navigation options either by using the hypertext capabilities of the program or by setting up navigation links [85].

When setting up navigation links, the author may choose from the following alternatives for a destination:

- Going to the next, previous, first, or last page in the *piece*.
- Going to a specific page in the *piece*.
- Allowing users to backtrack the path they have taken through *piece*.
- Allowing users to search for a particular topic in the *piece* and jump at the appropriate page.
- Allowing users to jump to a specific page.
- Using an expression to jump to a specific page.

In the past, Authorware *pieces* could be transformed into the Macromedia Shockwave [92] compressed file format by using the Afterburner companion application. Shockwave files could then be viewed using a suitable Shockwave web browser plug-in.

Afterburner compressed the files and divided the *piece* into suitable segments. Compression reduced the file size and therefore the transmission time. When the *piece* ran, the segments were downloaded to the user's computer only as required, rather than requiring the whole piece content to be downloaded at once. This approach reduced delays for users following the presentation flow.

Virtually all the features of Authorware [84] can be used when exporting to Shockwave [92]. Shockwave interactive *pieces* can include text, graphics, audio, animations, and movies, as well as all response types, data tracking, and database management. Shockwave *pieces* can be embedded within web pages or presented in full screen mode. It is possible to jump to any URL [196] from Shockwave *pieces*.

Current versions of Authorware have evolved to provide a companion Authorware Web Packager [93] application that replaces Afterburner. A matching Authorware Web Player plug-in is now required for web delivery.

The Authorware Web Packager [93] prepares the Authorware file for web delivery. It modifies the packaged Authorware [84] file — packaged without run-time — by converting it into smaller segments that can then be *streamed* from a server to the viewer's web browser.

The browser will begin to render the web packaged Authorware [84] *piece* immediately in the browser window rather than having to wait until the entire file has been downloaded. To manage all these segment files, the Web Packager [93] creates an editable map file, which dictates how and where to download the Authorware segment files. The size of these segment files can be modified inside the Web Packager [93] to match the Internet connection speed of the target user and expedite the download process appropriately.

The download specifications of all files that are external to the Authorware [84] *piece*, such as movies, Director [91] files, DLL's [152], XCMD's [79] and Xtras [94], will be directed by the map file as well. These external files are not converted to segment files but are downloaded in full as binary files.

2.6.3 Quest

The Quest [81] authoring system uses the *frame* paradigm as defined by Siglar [48]. Quest documents are called *titles*. Quest *titles* are composed of various *modules*. *Modules* contain *frames*, and *frames* contain the actual course content in the guise of various types of objects.

Quest provides two main creation and editing modes [82]:

- *Title Design*. This editing mode provides facilities similar to a flow chart or outline editor. *Modules* and *frames* can be visually organised and sequenced. A tree-view map of the entire *title* is also available for ease of navigation and quick reference.
- *Frame Edit*. The actual content for Quest *frames* is defined and structured using this editing mode, using the drag-and-drop mechanism and WYSIWYG technology.

A variety of other views are available — Info Map, Project View, preview windows, etc. — to assist the process of title creation and management.

Quest provides a number of productivity and re-usability-oriented features, such as:

- *FastTracks* let authors incorporate previously created objects, files, and groups into Quest titles. Media objects can be introduced into *FastTracks* by simply using the standard clipboard's copy and paste operations.
- *QuickFrames* facilitate the creation of multiple frames with a pre-defined structure. It is a mechanism that allows the definition of generic frame designs and their associated relationships, and it is very useful if certain unique frame designs would need to be shared among several titles or modules.
- The *Call Module* tool facilitates the invocation of a module from various locations within a title.
- The *Concurrent Module* tool can be used to develop facilities such as glossaries, help or navigation controls that may invoke other *modules*.
- Custom text styles can be defined and re-used throughout a *title*.

For extensibility purposes, Quest [81] includes an authoring language called Quest C, which is very similar to the C programming language. To alleviate the necessity to learn Quest C, the Quest C Coach tool allows beginners to select options through a wizard interface rather than requiring mandatory programming skills [82].

Quest supports Dynamic Data Exchange (DDE [153]) and ActiveX [154] integration, making it easy to embed titles into other types of documents, such as word-processing files, spreadsheets, and presentations.

Quest also supports direct extensibility, allowing integration with software components developed in other languages. Methods implemented in Windows DLL's [152] or executable files can be integrated into Quest and made available to Quest C scripts by using the *Extension Manager* tool [82].

The *Smart Spots Editor* facilitates the definition of complex, irregular visual regions [82]. This facility can be useful when defining visual maps for use with hyperlinks. Hyperlinks can also be associated with any words or groups of words.

The *Menu Editor* provides a convenient tool for creating menus that can contain submenus and commands implemented in Quest C scripts [82].

In terms of computer-based testing capabilities, Quest C supports the definition of typical interactions through buttons, active objects, drag and drop interfaces, text-entry fields, etc. User answers can be used for suitable feedback and scoring. For ease of development, the *FastTracks*

library contains pre-defined templates for designing true/false and multiple-choice questions, fill-in-the-blank questions and image-map tests.

When judging answers supplied through text-entry fields, Quest [81] provides a wizard that helps the author to define the evaluation criteria. The following types of answers may be supplied:

- Word or phrase Answers. In this case, the author specifies the exact word or phrase the student must supply for the correct answer. Allowances may be made for misspelling, case-sensitivity, etc.
- Numeric Answers. If the student should answer with a numeric value, either the exact number must be supplied, or a range of acceptable answers must be defined.

For playback, Quest [81] *titles* must be deployed with a helper stand-alone player application. Alternatively, they may be viewed using the *Quest Internet Player* [83] —realised as an ActiveX [154] control — for web-based delivery.

Quest also provides objects that facilitate peer-to-peer event and data handling over the Internet between users of Quest *titles*. This facility simplifies the development of interactions between *titles*, making it easy to orchestrate events and share data among *titles* used by various users connected to the Internet simultaneously.

Quest C offers an API for accessing arbitrary URL's [196], which offers a mechanism for dynamically updating or adding content to Quest [81] *titles* at runtime.

When using web-based delivery of Quest [81] *titles*, once the user has downloaded the main title file, the *title* software will take control of retrieving other associated media files — such as audio clips and graphics files — as specified by the author. As the author develops a *title* in Quest [81] with the intent to deliver it over the Internet, she can specify what caching policy is to be used for the various media files used by the title. Media files can be preloaded concurrently to enhance the user experience and they can be structured in groups of related files that can be safely removed when no longer required, or preserved for use later in the *title* to avoid having to download them again.

The *Internet FastTracks* library is a set of templates that provides ready-made templates for web-based Quest [81] *titles*. These templates contain pre-defined Quest C scripts that initiate HTTP [184] connections to retrieve media files required by the *title*, display an optional progress bar as the media files are downloaded and enforce the caching policy [82].

2.6.4 Assessment

The systems listed in this category have advanced multimedia-authoring capabilities, and also provide web-based course management support. They also provide extensive support for quizzes and tests.

2.6.4.1 Graphical annotation approach and capabilities

Graphics capabilities in ToolBook [96] are good, although without support for transparency and highlighting. Although animation is possible in ToolBook, it requires considerable effort.

The Authorware [85] concepts of *pieces*, *models* and *libraries* provide powerful mechanisms for structuring reusable presentation materials. The graphical capabilities of Authorware are very sophisticated, and can produce very high-quality graphics.

Graphical objects may be integrated into Quest [81] frames. However, there is no emphasis on dynamic presentation capabilities, unless the Quest C scripting language is used, which would require considerable programming efforts. Native support for transparency or highlights is not available.

2.6.4.2 Sound annotation approach and capabilities

Although sound annotations can be added to ToolBook [96] books and pages in the form of audio or video clips, they are not streamlined into the presentation process.

Authorware [85] is not geared towards streamlined voice annotation integration into pieces. However, sound annotations can be integrated, albeit in a rather laborious fashion.

Quest [81] is not geared towards sound integration into frames, the emphasis being placed on textual and graphical information. However, sound annotations are possible, if cumbersome.

2.6.4.3 Web integration approach and capabilities

For web delivery, ToolBook [96] course material may be delivered either as a package consisting of HTML [191] and Java [208] files or as native ToolBook [96] files rendered using a dedicated browser plug-in.

The Authorware Web Packager [93] application prepares Authorware [85] files for Web delivery by converting them into smaller segments that can then be *streamed*. A matching Authorware Web Player plug-in is required for web delivery.

Quest [81] titles must typically be deployed with a helper stand-alone player application. For integration into web pages, they may be viewed using the *Quest Internet Player* [83] —realised as an ActiveX [154] control.

2.6.4.4 Hyperlinking approach and capabilities

The *jumps* and *pop-ups* provided by ToolBook [96] provide good hyperlinking capabilities, either in the same window or using new windows. The *hot-words* mechanism for creating active areas takes a text-driven rather than a graphical approach.

The combination of graphics and text-based hyperlinks possible in Quest [81] resembles ToolBook [96], as previously illustrated in Section 2.6.1.

Authorware [85] provides a sophisticated branching framework for selecting paths through the presentation flow, as discussed in Section 2.6.2. Arbitrary URL [196] jumps are also possible.

Arbitrary URL [196] linking is possible, as is the ability to dynamically update or add content to Quest [81] *titles* at runtime.

2.6.4.5 Other distinguishing capabilities

ToolBook [96] provides the OpenScript scripting language in order to facilitate complex processing and programming [97]. Although this facility may provide a more intricate presentation experience, it requires more authoring time and expertise for arguably little extra presentation impact in most cases. Given the required programming expertise, ToolBook [96] may appeal more to *information technologists* rather than *instructional technologists*, as introduced in Section 1.3.2.

ToolBook [96] also provides facilities for developing quizzes. ToolBook [96] can also deliver presentations using the browser plug-in approach.

Authorware [85] provides for the creation of quizzes with sophisticated matching capabilities. Forms of scripting and advanced branching capabilities are also available.

The Quest C authoring language provides advanced authoring capabilities and the possibility of integration with virtually any application with public API's on the same platform by using the *Extension Manager* tool [82].

2.7 Summary

In this chapter, after briefly considering educational systems taxonomies to help focus our review and later assessment, we have examined the main authoring approaches for online lectures. We have then focused our attention on a number of web-based educational systems and assessed their comparative strengths and shortcomings.

We found that the multimedia authoring systems we reviewed generally do not provide a dynamic lecturing experience, their graphical animation, voice annotation and timing

capabilities being somewhat cumbersome to use. Support for high-quality graphical annotations, with transparency and highlighting support is also limited.

The graphics and voice authoring capabilities of web-based authoring systems are constrained by the development expertise required and time-consuming nature of the preparation and assembly of suitable HTML [190] resources.

On the other hand systems such as Authorware [85], ToolBook [96] or Quest [81] have extensive and very advanced authoring capabilities that may be worthy of emulation, and provide web-browser plug-in player components. They also offer a wide array of capabilities for the creation and assessment of quizzes and tests. However these systems do not place significant emphasis on bandwidth economy. For some solutions, such as Quest [81], significant expertise is required to configure caching policies in order to optimise Internet traffic.

Some systems, such as HyperCard [76], ToolBook [96] or Quest [81] provide comprehensive scripting capabilities in the pursuit of rich support for interactivity. However, effective use of this feature requires considerable programming expertise.

Driven by our primary research objective, in the next chapter we will introduce the main concepts, capabilities and characteristics of the AudioGraph web-based lecturing system, which were influenced by our observations described in this chapter.

3 The AudioGraph Web-based Lecturing System

In this chapter, we will introduce the AudioGraph web-based lecturing system. To begin with, we seek to help readers imagine the system's intended use in lecturing. To achieve this, we will explain the AudioGraph system's structure and briefly outline a typical usage example, which is illustrated in more detail in Appendix A.

We then develop our main requirements, guided by our research objectives, our analysis of the AudioGraph prototype [1] — as discussed in Section 1.3.4 — and informed by our assessments considered in Chapter 2. We present the main AudioGraph concepts that address our requirements, the web-ready file format and the authoring, playback and display models. We continue by outlining the history of the components of the AudioGraph system and clarify the contributions of the author in this context.

Subsequently, we review the main elements and capabilities of the AudioGraph recorder and player applications.

3.1 Use of the AudioGraph System in Lecturing

To describe the AudioGraph, we can draw a very interesting parallel between the underlying paradigm used by the AudioGraph system and the approach taken by Adobe for authoring and rendering Portable Document Format (PDF [199]) documents. In essence, for both systems, electronic media is created and rendered in a two-stage approach:

- An authoring application generates a device-independent description of the desired output in a suitable format. In the case of PDF [199], various authoring tools and API's — such as the Acrobat Distiller Server [201], Acrobat Distiller API [202], pdfmark [203] or the Acrobat PDFWriter API [204] — can be used to produce PDF files that contain graphical objects defined in a special page description language. In the AudioGraph system, the AudioGraph recorder can generate episode files that conform to the AudioGraph web-ready file format specification.
- A playback program controlling a specific output device interprets the description and renders it on that device. In the case of PDF [199], tools such as Adobe Reader [200] can be used to view or print PDF files. In the AudioGraph system, the standalone player or the web-browser plug-ins can be used to present AudioGraph episodes.

These two stages may be executed in different places, on different platforms and at different times; the specific file formats serve as interchange standards for the compact, device-independent transmission and storage of documents or educational multimedia content.

The AudioGraph system consists of the lecture recorder and the players, which may be either web-browser plug-ins or standalone applications, as shown in Figure 5.17.

As the main users of the AudioGraph lecture recorder, educators may create lectures, either from scratch, or by importing collections of images — which may represent slides from existing material created with various forms of presentation or text-editing software. The simple approach and familiar user interface of the AudioGraph recorder may facilitate the inexpensive creation of multimedia web-based lecture material.

Pearson and Jesshope estimated that the typical effort required to prepare an AudioGraph lecture takes between two and three times the time needed to present it once to a class, and it is likely to drop over time [5]. The main benefit stems from the availability and multiple use opportunities. The time freed up from synchronous lectures may be used for more tutorials, research or seminars.

When editing lectures, authors may either edit slides (of a defined rectangular size) or scrolls (with a defined width and of virtually indefinite height, over which the perspective may scroll as directed by the author), and in the process they would typically create and manipulate annotations as well, such as voice, image or graphical components. Authors may also export lectures, thus readying them for web-delivery. Once available in this format, they may be published on web servers or integrated in wider learning management systems.

Students may then independently view the lectures in their own time, at their own pace, using an AudioGraph player, either as a stand-alone application or through a web-browser plug-in. Students could be freed to pursue more in-depth studies or cover more ground as they have the opportunity to make better use of their time by learning at their own individual pace, skimming or skipping familiar subjects and dwelling on new, unfamiliar or interesting topics.

Appendix A provides an illustrated, systematic guide to using the AudioGraph system for authoring and presenting lectures. The process described spans the creation of an AudioGraph lecture, through its editing and export, to playing it back in a web browser, using version 1.0 of the AudioGraph system developed on the Apple Macintosh platform.

3.2 Requirements Analysis

In this section, we formulate our main requirements, guided by our research objectives, our analysis of the AudioGraph prototype [1] — as discussed in Section 1.3.4 — and informed by our assessments considered in Chapter 2.

3.2.1 Requirements Derived from the Analysis of the AudioGraph Prototype

This category of requirements arose from the aspiration to mitigate the shortcomings of the AudioGraph prototype, which we discussed in Section 1.3.4.

Rather than focusing the recording experience on individual *frames*, we considered it would be more intuitive for our target users, lecturers, if we started from the concept of *lecture*, which would contain a number of *episodes*. An *episode* would be considered the equivalent of a transparency slide. We envisaged that the sequence of *episodes* in a *lecture* should be organised through simple drag-and-drop or cut and paste operations, facilitating lecture preparation, just as slides can be stacked in the right order prior to lecture delivery.

In our context, the elements called *frames* in [1] would be called *episodes* from now on. Similarly, what the prototype [1] referred to as *events*, would be named *annotations* or *components* in the AudioGraph. When preparing a web-based presentation, the *lecture*, or a user-defined selection of *episodes* from the *lecture*, would be exported in web-ready format, eliminating the need to edit any HTML [190] files at all, since their sequence could be correctly defined in advance in the *lecture*.

We considered prototype's inability to support individual annotation selection and editing a severe limitation and sought to remedy the situation by adopting an *iconic* representation for the presentation stream, as later categorised by Siglar [48], which could reflect annotation types and their associated attributes.

The prototype made no use of transparency for its images or graphical annotations. We considered that fine-grained control over annotation transparency and erasing effect could facilitate more expressiveness for advanced users.

In the prototype, the authoring tools imbued annotations with their attributes, which became immutable. In contrast, we sought to have the colour, pen size, transparency, stealth, highlight and erasing attributes editable for each graphical annotation at any point in time.

The range of colours in the prototype was drastically limited to just six — black, red, blue, green, magenta and cyan. In contrast, for the AudioGraph system, we sought to offer users the flexibility of choosing from the full palette of millions of colours available in a 24-bit RGB colour space.

The prototype chose to use single images as the backgrounds for its *frames*. We reasoned that although the concept of background image might be retained, multiple images should be available in *episodes* for display or animation. It should be possible to display and hide images at will, zoom images in or out, crop or stretch them to fit given rectangles. Images should also

be capable of transparency, and a better image format than GIF [225] and preferably royalty-free would be required.

The sound quality in the AudioGraph had to improve over what the prototype had to offer. We needed to find a compression scheme optimised for voice, with better quality and lower bandwidth requirements.

Consistent with good human interface design principles, as discussed in Section 4.1.1.8, in order to provide forgiveness for its interface, the AudioGraph had to implement multiple undo and redo capability, just as the prototype presented in [1] had done.

3.2.2 Requirements Derived from the Analysis of Existing Lecturing Systems

This category of requirements was derived from the desire to emulate a number of the valuable characteristics of existing lecturing systems, avoid their shortcomings as best as possible and provide some useful capabilities not widely supported by them, as discussed in Chapter 2.

It should be noted that in contrast to systems such as Authorware [85], ToolBook [96] or Quest [81], the AudioGraph did not seek to provide explicit support for tests and quizzes. In light of our main research objective, we purposefully chose to focus on the lecturing rather than the evaluation aspects of the educational experience.

At times, lecturers may also use a transparency film that can be scrolled over the projector. We sought to capture this presentation approach by providing two types of *episode*: the *slide* and the *scroll*. *Slides* would have a defined width and height, which would govern the size of the web browser plug-in display area, and would be immutable. *Scrolls*, on the other hand, would have a display area of virtually unlimited height, over which the current view could be scrolled, replicating the scrolling transparency experience.

In order to facilitate animation of graphics, we envisaged it would prove useful if groups of graphical annotations could be displayed at once, providing a sharp switch between animation frames. To address this requirement, we designed an attribute for graphical annotations, called *stealth*, which would cause their display to be delayed until the first non-stealthy annotation was encountered in the playback stream.

Another useful extension we envisaged would be that of annotating episode *components* with text comments — which could represent the transcript for audio patches, and it might even be used as the source for computer-synthesised speech.

The image compression to be used should be optimised for best results with monochrome text and relatively few colours for backgrounds, similar to what most lecturers seem to use.

If authors seek to produce crisper than hand-drawn text, then text annotations, using any font the author desires, could also take advantage of this image format to produce high-quality, anti-aliased text representations with modest bandwidth requirements.

For ease of editing many annotations, it should be possible to bundle annotations into groups and handle them more conveniently in aggregated form.

3.3 The AudioGraph Concepts

Driven by the motivating factors outlined in Section 1.1, and taking into consideration the assessment of existing lecturing systems and the analysis of our main conceptual and technical requirements, we have set about creating the AudioGraph system. In the following sections we will explore its main elements: the AudioGraph web-ready file format and its associated authoring, playback and display models.

3.3.1 The AudioGraph Web-ready File Format

The first focus of our efforts has been the crystallisation of the version 1.0 of the AudioGraph web-ready file format specification. This file format definition was derived from the work on the early AudioGraph prototype [1], and it contained essential enhancements such as support for transparency, arbitrary numbers of image annotations rather than a single background image, and the concepts of image definition annotations and multiple audio streams that can help in reducing bandwidth requirements.

The file format represents *episodes* as a sequence of records. Some records apply to the entire episode and others are related to specific components. Images and sound annotations make interesting examples.

For images, a potential option might be to store image data within the record that also triggers the image display. However, this approach would waste considerable bandwidth if the same image would need to be displayed in more than a single location or if it would be animated. Therefore, image data is stored in a special kind of record that has no visible effect, and which adds the image to an internal dictionary. Separate image display records refer to the dictionary and indicate whether the image should be zoomed, stretched or cropped to the given display region rectangle.

A similar approach, in reverse, is taken with the sound annotations, by defining sound stream characteristics — compression method, sample size, sample rate, etc. — in a special kind of unique record, and then having sound data records that refer simply to the particular stream rather than having to carry lots of configuration information with each sound patch.

Since commercial interests related to the AudioGraph system [3] are currently at stake, we would prefer not to disclose the exact file format specifications. We hope that the above descriptions give sufficient indication of the general approach to form an impression of the issues involved in its design.

In order to validate the 1.0 AudioGraph web-ready file format specification, the author has designed and implemented the Macintosh 1.0 AudioGraph recorder, web-browser plug-in and the standalone player applications, and has created the Windows browser plug-in to support this specification as well, based on an early, unfinished skeleton begun at Surrey University.

As the AudioGraph system has matured, new capabilities have been introduced, such as linked episodes and support for arbitrary hyperlinks, clearscreen annotations, and other enhancements, which have been captured in version 2.0 of the AudioGraph web-ready file format specification.

The author has been actively involved in the definition of the 2.0 AudioGraph web file format specification, and has designed and developed the Macintosh 2.0 web-browser plug-in. It features a radically improved rendering engine, that led to substantial performance improvements, and implements the specification nearly completely — the only missing feature at the time of writing was support for the Ogg Vorbis [242] sound format.

In the following sections, we will examine the AudioGraph authoring, playback and display models assumed by the web-ready file format specification, and consider the various differences between versions 1.0 and 2.0.

3.3.2 The AudioGraph Authoring and Playback Model

Given that the overriding goal of this research has been to produce a system that is readily useable by any lecturer with minimal training in creating multimedia content, the most important value we have strived for has been simplicity.

As a result, from a conceptual perspective, we have chosen to maintain the single event stream concept introduced in the prototype [1]. The author creates a lecture, adds an episode — perhaps a slide — and sequentially adds annotations, either graphical or aural. During playback, the sequence of episodes and episode annotations is reproduced in the exact order in which it was recorded by the author, with no provision for concurrency.

Although no parallelism or synchronicity of playback over multiple timelines is explicitly provided for, we can endow visual components with an attribute of *stealth* that makes sequences of such components display as a single component during playback the moment an aural or a non-stealthy component is encountered. This facility is useful in creating complex, smooth

animations by allowing us to define full frames that are drawn at once, resulting in smooth frame transitions.

3.3.3 The AudioGraph Display Model

The AudioGraph players are required to use the display structure shown in Figure 3.1.

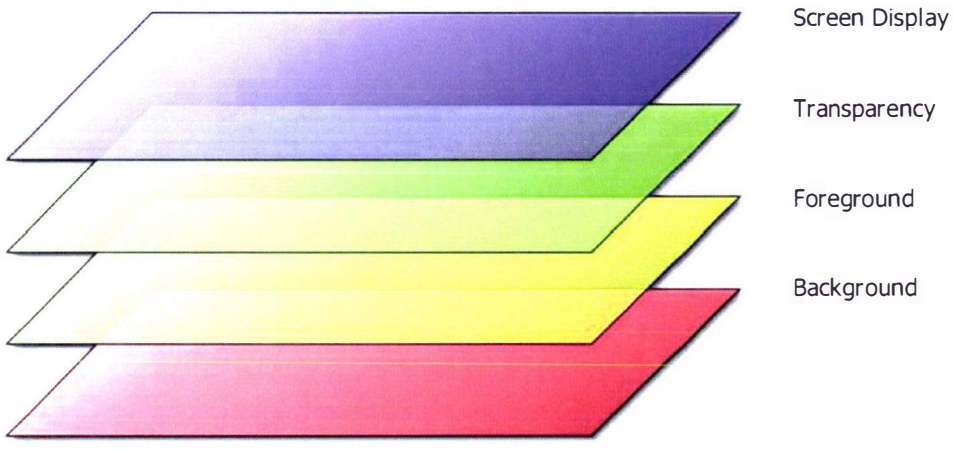


Figure 3.1. The AudioGraph display model

The *screen display* layer represents the image visible to the student at any one time. The *transparency* layer — also called *alpha channel* layer — is used to record the transparency of the foreground layer[1]. The *foreground* layer captures all graphical annotations apart from images. The *background* layer is coloured uniformly with a background colour, and it currently captures all the effect of the image annotations.

In this context, all graphical annotations are superimposed over any images in the presentation. Although it would be possible to use foreground images, particularly for representing easily erasable text, this capability has not been specified or implemented to date.

The screen, foreground and background layers record pixel information as colour values, encoded either as RGB triplets or as colour table indexes when using an indexed display mode.

The alpha channel layer always records transparency values in an unsigned byte range (0~255), where zero means the foreground is opaque and 255 means the foreground is fully transparent. The initial state of the alpha channel layer is fully transparent. Therefore, the initial values in the foreground layer are unimportant.

The background layer is initially white, and then becomes filled with the background colour specified in the appropriate header record, as documented in the AudioGraph file format specification.

---

[1] It is interesting to note that the latest version of the PDF [199] specification — 1.4 — uses a similar approach to the one adopted earlier by the AudioGraph system, by using a transparent imaging model.

All graphical annotations share a number of primary attributes: colour, pen size, and, in the 2.0 version AudioGraph web-ready file format specification, line end form.

3.3.4 The AudioGraph Annotations

In this section, we examine the different types of annotations supported by the AudioGraph.

3.3.4.1 Normal Visual Annotations

The web-ready file format requires only the implementation of a pen annotation, as all graphical annotations can be drawn in terms of pen operations, as if tracing a virtual pen over the episode canvas.

A simplifying assumption that is currently manifest in the visual model is that visual annotations have uniform solid colour, although they may have a transparency value associated as well. In the future, patterns or gradients may be considered as well, as they are present in many widely available graphical authoring applications.

The pen size refers to the width of the lines marked with the virtual pen. In version 2.0 of the AudioGraph specification, lines can be marked as rounded, as seen in Figure 3.2.

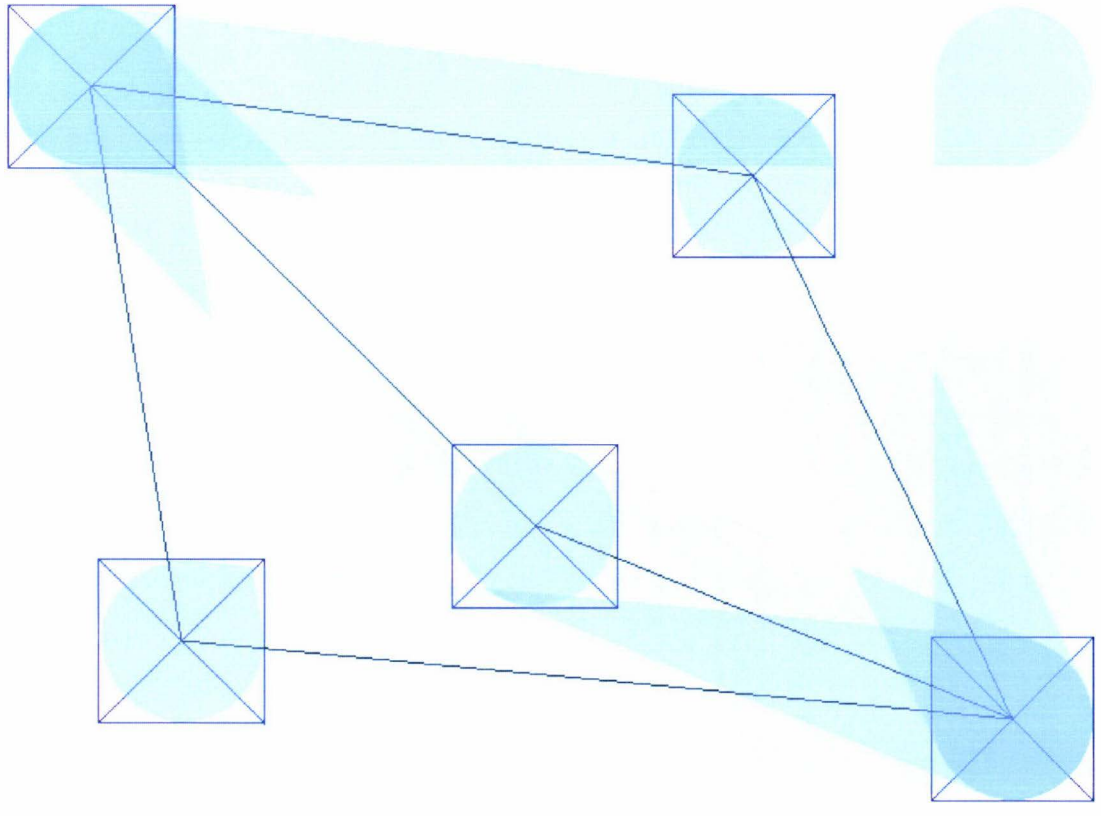


Figure 3.2. Round line drawing examples.

In contrast, version 1.0 of the AudioGraph recorder assumed all lines would be sharp, using rectangular pen tips, and interpreted the pen size value to mean the width and height of the pen tip rectangle rather than the actual line width.

In Figure 3.3, sharp lines of identical width are shown using the same coordinates as the round lines in Figure 3.2. Notice that the round lines have ends that superimpose perfectly for the same line width, regardless of direction. By comparison, sharp lines have different line end geometries, as dictated by the approximations required to obtain the same line width regardless of the line direction and the additional constraint that the sharp line end must end in a rectangle whose centre is to be situated at the line extremity coordinate.

Drawing a normal visual annotation requires a number of steps:

- The first one is to recalculate — blending if transparency is used — the colour and the transparency of each affected pixel in the foreground and transparency layers. No action is taken if the annotation is fully transparent.
- Second, the colours of the corresponding pixels in the screen layer are calculated using the colours of the pixels in the background, foreground and transparency layers, blending them according to the transparency of the foreground as recorded in the alpha channel layer.

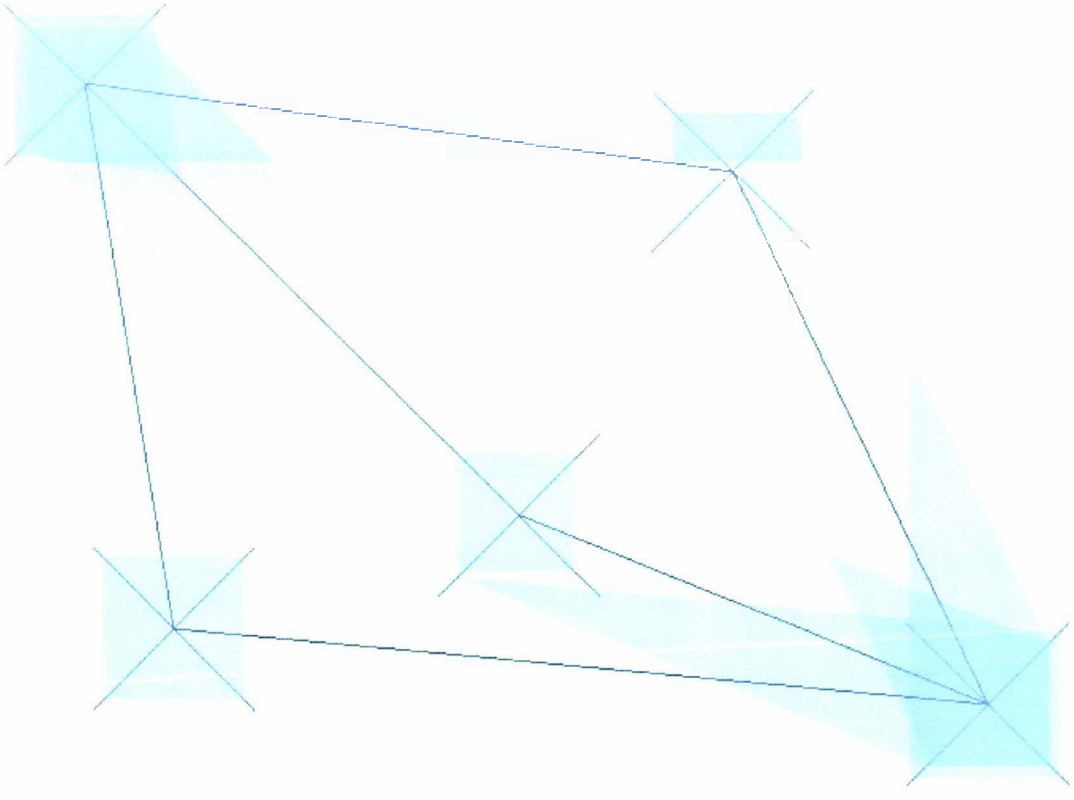


Figure 3.3. Sharp line drawing examples

For each colour channel — red, green, and blue — the following equation stands:

$$\mathbf{screen = alpha * background + (1-alpha) * foreground}$$

Where **alpha** is normalised ($\mathbf{0 \leq alpha \leq 1}$), zero giving opaque visual annotations and a value of one leading to fully transparent annotations.

3.3.4.2 Erasing Visual Annotations

The AudioGraph visual model also allows graphical components to have an erasing effect. Such components affect only the transparency of the foreground layer, and therefore the colour of these annotations is irrelevant for their visual effect.

For erasing annotations, the new transparency value recorded in the alpha channel layer is calculated as follows:

$$\mathbf{alpha_{new} = 1 - annotation\_alpha * (1-alpha)}$$

Where $\mathbf{alpha_{new}}$ is the new transparency, **alpha** is the old transparency value, and **annotation_alpha** is the transparency value specified in the erasing component. All transparency values are normalised in the above equation ($\mathbf{0 \leq alpha \leq 1}$ and $\mathbf{0 \leq annotation\_alpha \leq 1}$).

Note that opaque erasing components — with **annotation_alpha** zero — render the foreground totally transparent, while fully transparent erasing components — using **annotation_alpha** one — have no visible effect.

3.3.4.3 Highlight Visual Annotations

Another feature of the AudioGraph visual model worthy of attention is the use of *highlight* components. These components are drawn directly in the screen display layer, superimposed over the foreground and background layers, using a bit-wise **and** of the screen layer colour and the highlight colour.

An uninterrupted sequence of such components is considered to be a *highlight group*. It continues to affect the screen layer until a new highlight group takes its place, when the previous highlight group vanishes. This approach allows simple and elegant use of highlighting to direct the focus of attention to designated areas while audio explanations are underway.

Note that since the binary representation of the colour white contains only bits set to one, white highlights have no visible effect — apart form hiding any previous highlight group, for which purpose they can be safely used.

3.3.4.4 Image Annotations

Images drawn in the AudioGraph display model only affect the background layer. Even so, the AudioGraph fully supports transparency, and therefore images with transparent areas produce the expected results, blending with the background or overlapped images. The erasing attribute is also extended to image components, although with a slightly modified semantic. The visual effect of erasing image annotations is to make a corresponding image component 'vanish'. In other words, as soon as a 'vanishing' image component is 'displayed', the nominated normal image component 'disappears', the background and screen layers being restored to a state equivalent to the component never having been displayed there.

Image rendering in the AudioGraph playback model uses a two-step process:

1. Image definition annotations that have no visual effect apart from defining the source image data must first be available in the annotation stream.
2. Images are rendered by small image manipulation annotations — contributing to bandwidth economy — that define the display location and clarify whether picture zooming, clipping or stretching is required.

This approach has the benefit of reducing bandwidth requirements significantly when images are used for animations or are replicated multiple times, as the same image definition can be reused repeatedly with different visual attributes specified by image manipulation annotations. Such annotations are extremely small compared to the average image file size. For this reason, even when image data is used only once, the bandwidth penalty for using this approach is virtually nonexistent.

Given our target audience — lecturers — we estimated that that many of the images used in AudioGraph episodes might contain plain text in case users would prefer not to use their digitised handwriting for dynamic annotations. Therefore, we envisaged a text tool for the AudioGraph recorder, which although the author did not implement in version 1.0, was eventually fully implemented in version 2.0 of the AudioGraph. The text tool generates an optimised image representation of the anti-aliased text annotation, as shown in Figure 4.6, with a minimal colour and transparency gradient palette for excellent bandwidth economy [15].

In this context it might have been argued that for such images, the compression ratio might have been improved even further beyond what optimised loss-less image compression formats — such as PNG [220] or GIF [225] — would be capable of, should we have considered the approach of storing the text itself in display annotations and the font properties in unique, per-episode annotations, adopting an approach similar to the one we used for sound streams.

However, this approach is not easily portable when oblivious of native fonts unless we would resort to a more complex rendering mechanism, as used by the Adobe Portable Document Format (PDF [199]) file format, in which case we have probably compromised the bandwidth economy restrictions as we would have had to embed fonts or font character subsets.

3.3.4.5 Scroll Annotations

In order to mimic another presentation approach sometimes used by lecturers, the AudioGraph model provides the concept of *scroll*. Just as lecturers may use scrolling transparencies for overhead projectors or whiteboards and slide the image up and down or across, the AudioGraph model allows the author to 'scroll' the presentation across the rectangular display area.

Conceptually, the author can use an almost unlimited canvas for presenting a scrolling episode. The only limitation is that we have chosen to use 32 bits to represent coordinates and therefore, the canvas must be limited to a rectangle of maximum 2^{32} pixels width and height. This limit is never reached in practice, as it would be equivalent to over four million average screen widths — considered to be slightly above one thousand pixels currently.

The scroll annotation simply indicates which point is to be the new top-left corner of the presentation. Using this simple mechanism, the screen image could be scrolled as desired, in any direction on the presentation canvas. However, the recorder application currently only allows for vertical scrolling, mimicking precisely the behaviour of transparency scrolls.

3.3.4.6 Clearscreen Annotations

Given the layered architecture of the visual model, it is sometimes useful during an episode to be able to wipe the presentation canvas clean. To cancel the visual effect of the foreground and transparency layers, a single opaque erasing filled rectangle as large as the display area would suffice. However, if multiple pictures were visible in the background layer, then having to erase each picture would be cumbersome and inefficient.

To address this deficiency and pave the way for presenting multiple episodes in the same plug-in player instance, the *clearscreen* concept was introduced in version 2.0 of the AudioGraph web-ready file format. A clearscreen annotation makes the alpha channel layer transparent and specifies a colour for the background layer to be filled with. As a result, it has the effect of clearing the presentation of any images, annotations or active links. Each episode can be said to have an implicit clearscreen as the first visual component, which sets the stage for the episode to unfold. As episodes are linked in the same presentation window, the implicit clearscreen annotations clear the visual layers in between episodes in order to avoid confusion.

For advanced authoring requirements, image manipulation annotations can also be marked as *persistent*. This feature is useful when linking episodes in the same presentation window and the author wishes to preserve certain images between episode boundaries. Persistent images are the only annotations that continue to be displayed after implicit or explicit clearscreen annotations have wiped clean the background, foreground and transparency layers.

3.3.4.7 Sound Annotations

There can be several virtual *audio streams* in an episode. Each audio stream has its own properties: compression method, sampling rate, sample size, etc. used in determining the appropriate decompression and playback methods for all the audio annotations belonging to the stream. This approach minimises the data required to encode audio annotations, while preserving flexibility in the choice of compression and therefore sound quality. The alternative, mandating that all audio annotations be encoded using the same method and therefore eliminating the need for audio stream definitions would have been too restrictive and inflexible and the gain in reduction of data requirements by eliminating the audio stream descriptors would have been minimal.

Sound annotations indicate which audio stream they belong to and provide the sound patch data in the format indicated by that stream. The playback engine decompresses and processes the sound data as indicated by the audio stream descriptor.

It is possible to circumvent the synchronicity limitation to a certain degree as sounds can be played asynchronously, effectively providing voice-over capability while the presentation may be playing back an animation sequence.

3.3.4.8 Pause Annotations

In order to provide fine control over animation and the playback experience in general, pause annotations are available. Although pause durations can be specified by the author in the AudioGraph recorder in increments of $1/10^{th}$ of a second, the file format allows for durations expressed in milliseconds. It is interesting to note that in version 2.0 of the AudioGraph file format specification, all visual components that are not marked with the stealthy attribute are followed by an *implicit* pause. The length of this special, implicit pause can be controlled in the web-ready file format, and might also be user controllable in future versions of the players.

Each player has a default implicit pause duration set at manufacture to define the playback speed for an average computer. Currently, the default implicit pause duration is set at $1/60^{th}$ of a second on the Macintosh platform, also known as one *tick*. The user might override the default playback speed to suit her specific computer. However, if a special playback speed component

would be encountered in an episode, it would override the user setting for the duration of the episode, having the highest priority.

3.3.4.9 Link Annotations

The introduction of link annotations in version 2.0 of the AudioGraph file format has marked a significant advancement in the AudioGraph presentation paradigm. Previously, AudioGraph episodes were stand-alone presentations. They could be examined one at a time, but there was no means of providing in-line references to other episodes or resources.

Link annotations provide an alternative means of exploring information, by providing a mechanism for accessing various other resources as specified in the episode content. Links can reference other episodes, or arbitrary URL's [196]. In the case of the AudioGraph browser plug-in, the targets of link annotations can be accessed in the same plug-in instance, the same browser window, or in a new browser window.

Link annotations have no direct effect. They are intended to adorn the following annotation, typically a pen or an image manipulation component. Subsequently, when the adorned visual annotation is displayed and the presentation is stopped, the link becomes active and the cursor changes from the default arrow to a pointing hand when it hovers over the visual area of the adorned annotation. Naturally, clicking the window in this situation will instruct the plug-in to follow the link.

It is also possible to mark links as automatic. Such links will trigger the download action as soon as the adorned annotation is played. This mechanism can facilitate the construction of presentations that will loop indefinitely.

3.3.4.10 Halt Annotations

Another useful feature included in the 2.0 version of the AudioGraph file format is the halt annotation. Its effect is to stop the presentation playback, and it has no visual effect.

Such annotations can be used in a variety of ways, and they are particularly effective when used in conjunction with link annotations.

3.4 Evolution of the AudioGraph System Components

The AudioGraph system has evolved through a number of versions, with varying contributors and capabilities.

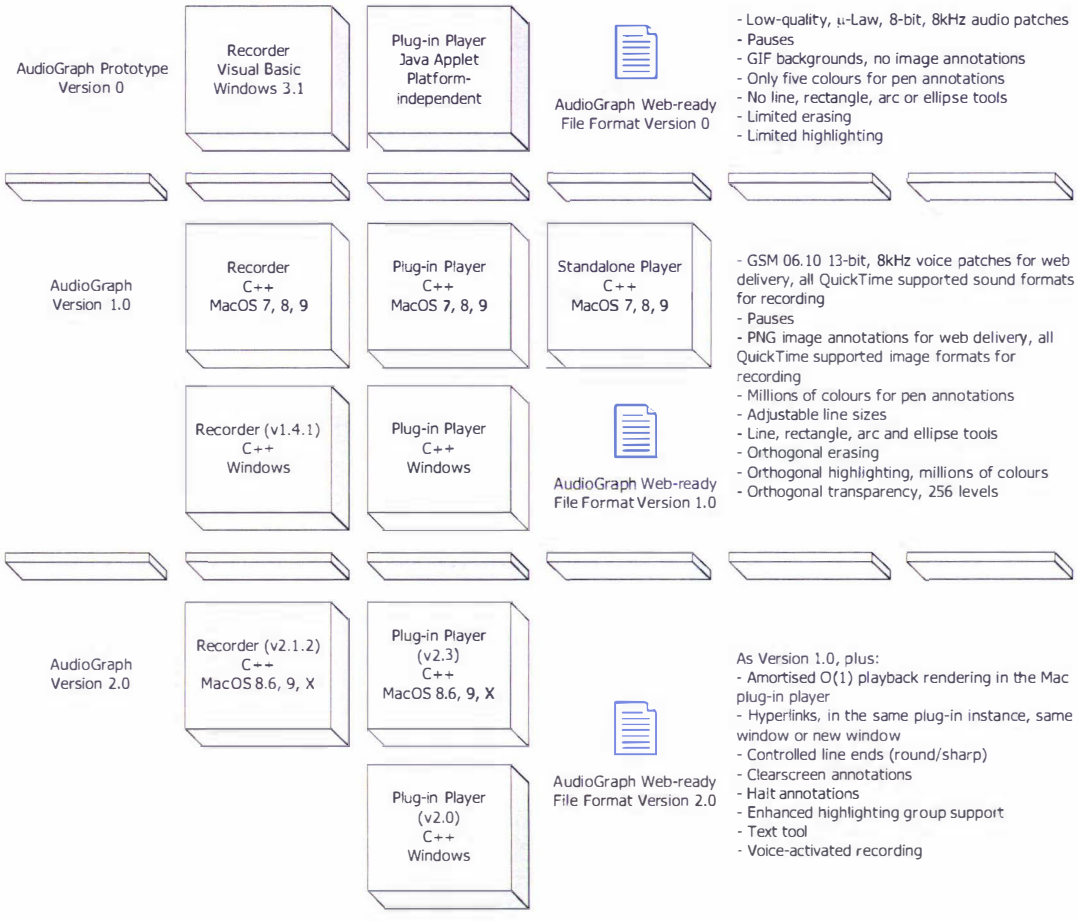


Figure 3.4 AudioGraph system evolution

The AudioGraph prototype refers to the experimental applications developed at Surrey University primarily by Dima Barsky and described by Jesshope and Shafarenko [1].

The boxes with a gradient background pattern in Figure 3.4 — the version 1.0 Macintosh applications, the version 1.0 Windows plug-in player and the Macintosh plug-in player conforming to version 2.0 of the AudioGraph web-ready file format — indicate the components created by the author. In addition, the author has contributed to the elaboration of versions 1.0 and 2.0 of the web-ready file format specifications. Although originally based on an unfinished skeleton begun by Dima Barsky at Surrey University, version 1.0 of the Windows plug-in player evolved considerably through the author's work and eventually fully implemented the 1.0 web-ready file format specification.

Where point versions are given in brackets, such as Plug-in Player (v2.3), they are the latest ones currently available from NZEdSoft [3], as of January 2004.

The Windows recorder (v1.4.1) has been a relatively recent effort to which the author did not contribute beyond a certain amount of guidance and advice. Although it was developed in roughly the same period of time as version 2.0 of the Macintosh recorder and plug-ins, it effectively provides a virtually identical user interface and functionality to the version 1.0 Macintosh recorder.

3.5 The AudioGraph Recorder

One of the crucial initial decisions in our work was to pursue its practical aspects using Apple Macintosh computers and technology, in part due to their prevalence in educational institutions but most importantly for the ease of multimedia manipulation.

Conceptually, the AudioGraph recorder manipulates *lecture documents*. Typically, lectures are composed of one or more *episodes*. Similarly, episodes contain a sequence of one or more *annotations* or *components* of various types, with either visual or aural effects. Although both lectures and episodes may be completely empty, without any content whatsoever, this is hardly ever useful in practice.

In designing the graphical user interface for the AudioGraph recorder application, besides considering general human interface design principles, given the fact that the Macintosh platform was our primary target, we have sought to conform to the Apple Human Interface Guidelines [158], as discussed in more detail in Section 4.1. In the following paragraphs, we will examine the user interface as defined in the 1.0 version of the AudioGraph recorder, as the author has created it.

3.5.1 Lecture Documents

The preparation of AudioGraph lecture documents can be initiated in a number of ways. They may be edited from scratch, or generated by importing Macintosh Scrapbook files containing images. The choice of Macintosh Scrapbook file as an intermediate format is intentional, as PowerPoint [64] presentations can be exported in this format, and any other document that can be printed using the standard Macintosh interface can be transformed into this format by using the Print2Pict 3.7.1 virtual printer driver, which can be downloaded from various sources [162]. In addition, Scrapbook files can be created manually with pictures from any sources, giving the user maximum flexibility in assembling a lecture skeleton.

There are two alternative views for the list of episodes: by *titles* and by *thumbnails*. Figure 3.5 reflects the *titles* view. In this view, the episode titles and their estimated durations are listed in a table. Notice that to the left of the titles, small icons indicate whether the episodes are slides or scrolls, and whether they have been modified or not — indicated by the small red lines.

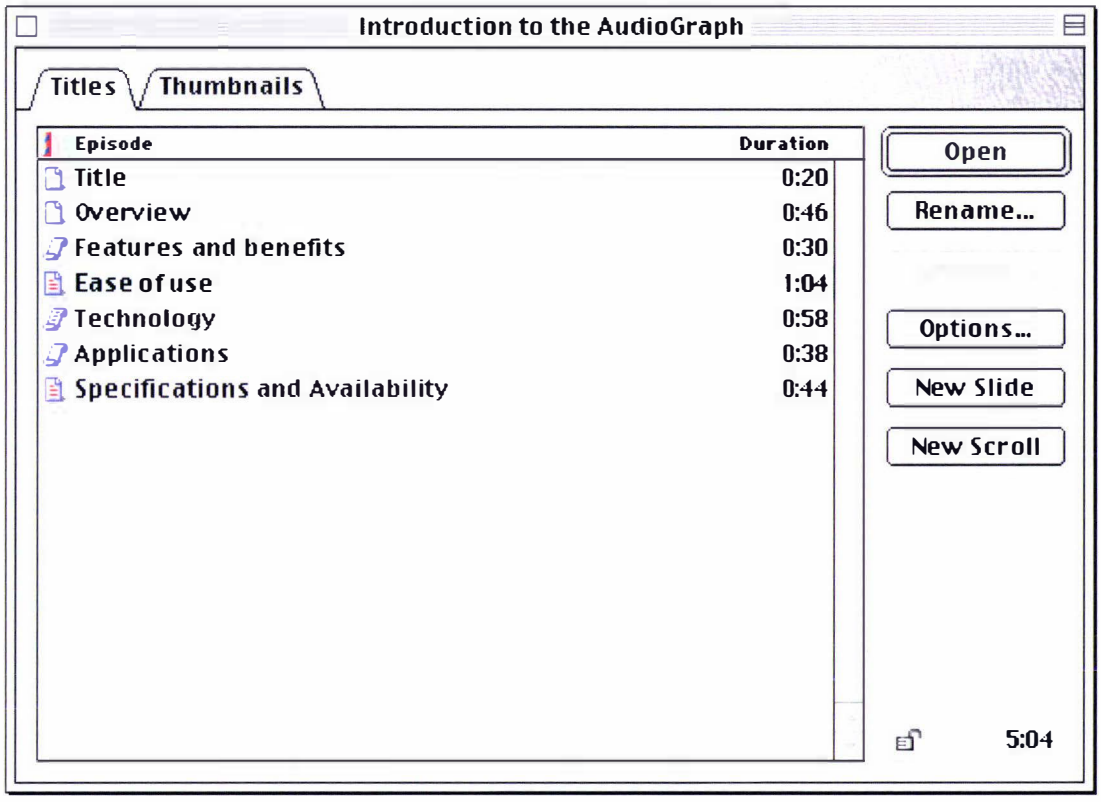


Figure 3.5. Typical lecture window for version 1.0 of the AudioGraph recorder

The thumbnails view, illustrated in Figure 3.6, presents thumbnail representations of the episode backgrounds:

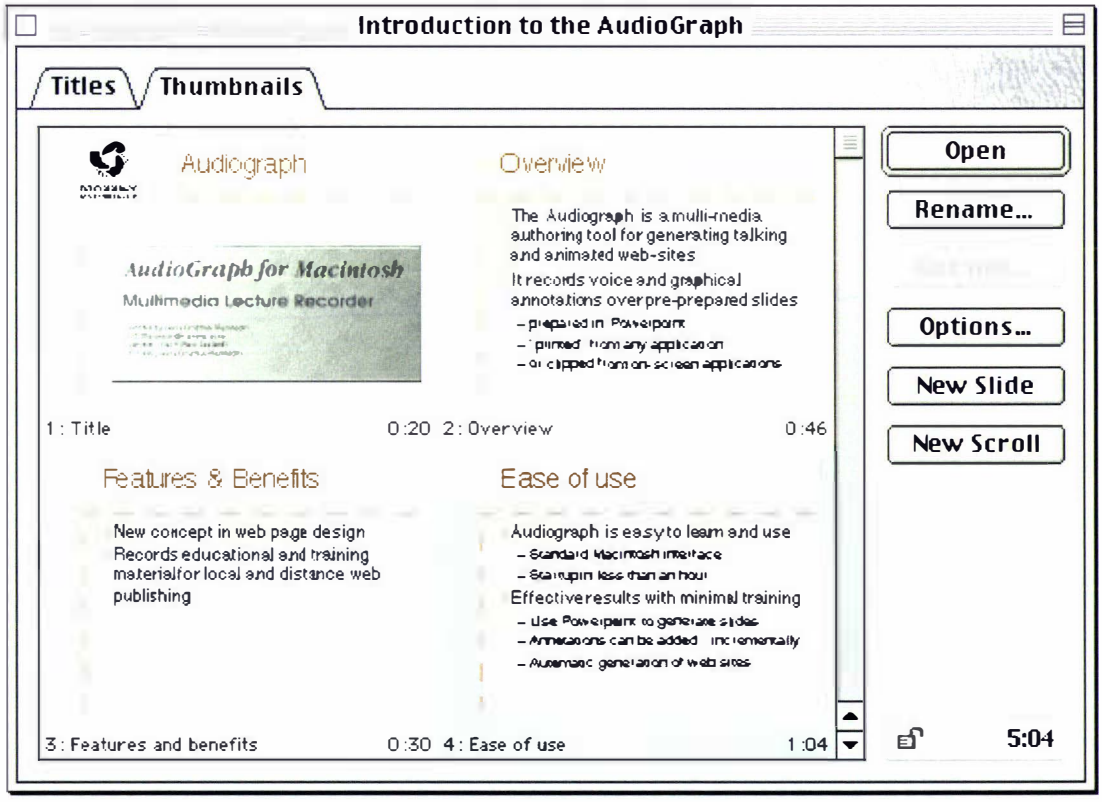


Figure 3.6. Thumbnails view for version 1.0 of the AudioGraph recorder

Lectures are saved in a custom file format that can preserve all forms of high-quality sound and image data supported by QuickTime [228]. Using this approach, when future versions of the AudioGraph will support other export file formats other than PNG [218] and GSM 06.10 [227], then the highest quality transcoding can be ensured.

Lectures can also be exported in the web-ready file format in two modes: *compact* and *expanded*. In the *compact* mode, all episodes in the lecture will be generated in the same web-ready episode file. In the *expanded* mode, an index web page will be created and each episode will be generated in its own file, making web streaming more efficient.

3.5.2 Episodes

Individual episodes are edited by opening up their own individual editing windows. Notice that if the lecture document is locked, the editing tools will be disabled when viewing episodes.

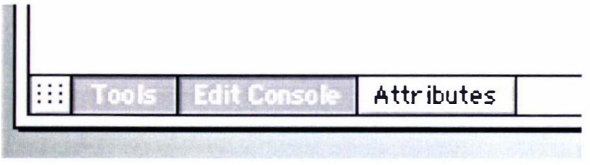


Figure 3.7. Episode window controls

The episode window provides an exact visual representation — what you see is what you get; see Section 4.1.1.5 — of the episode and its associated annotations. This window has a small number of controls, shown in Figure 3.7, that allow the user to specify which floating palette should be visible and whether all graphical annotations should be made to snap to a conceptual 'grid' or not.

As in most image editing packages, floating palettes assist the main edit window to provide editing tools and other controls. The intention is to provide users with a familiar authoring experience and controls that provide functionality according to normal expectations similar to other widely available graphics authoring tools.

In the AudioGraph recorder, three kinds of floating windows are available: the *tools* palette, the *edit console* and the *attributes* palette. We will examine the functionality provided by the controls in these windows in the following sections.

3.5.2.1 Tools Palette

The tools palette window, shown in Figure 3.8, provides the main authoring tools available for annotating episodes.

The *pen* tool allows the author to perform free-hand drawing. Although the AudioGraph recorder works with any pointing device, using a pen-based device makes it feasible for the author to annotate episodes with handwritten text, much as she would annotate transparencies.

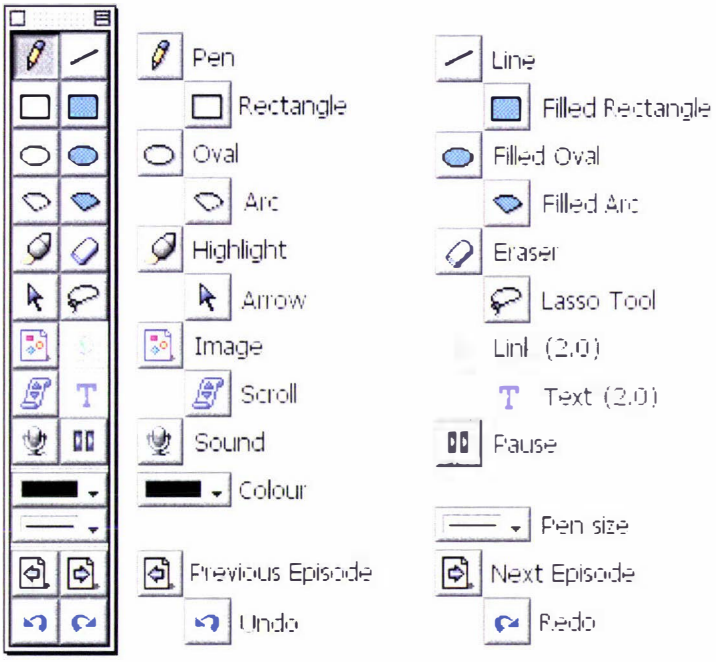


Figure 3.8. The tools palette and its controls for version 1.0 of the AudioGraph recorder

Therefore, pen-based input devices are recommended for lecturers that wish to make use of handwritten annotations. The ability to use handwritten annotations also suits Asian scripts very well. In version 2.0 of the system, by default, the annotations created with the pen tool have round ends.

The *line* tool offers the ability to draw lines. Pressing the *shift* key constrains the line to vertical, horizontal or diagonal, depending on the cursor position. In version 2.0 of the system, by default, annotations created with the line tool have sharp ends.

The *rectangle* and *filled rectangle* tools simplify the drawing of rectangles. Pressing the *shift* key constrains the rectangles to squares.

The *oval* and *filled oval* tools can be used to draw ovals. Pressing the *shift* key constrains the ovals to circles.

The *arc* and *filled arc* tools are available for drawing arcs. Drawing ovals and choosing a pair of start and end angles defines arcs. As with the oval tools, pressing the *shift* key constrains the ovals to circles. Ovals are a special case of arcs, where the start and end angles are equal.

The *highlight* tool simplifies the task of drawing highlights. It produces filled rectangles with the highlight attribute. By default, selecting this tool will change the current pen colour to yellow, as yellow highlights are most common. Reverting to any other drawing tool will restore

the previously selected colour. However, the colour used by the highlight tool can be changed as required, and it will be cached for the duration of the editing session. In other words, selecting the highlight tool again after using some other drawing tool will restore the latest selected highlight colour. The Attributes palette must be used if the author wishes to mark different kinds of annotations with the *highlight* attribute.

The *eraser* tool provides a convenient means of erasing annotations in the foreground layer, just like an erasing pen or an eraser. The pen colour control becomes disabled when selecting this tool, since the colour attribute is unused by the erasing operation. The Attributes palette must be used when the author wishes to mark different kinds of annotations with the erasing attribute, or if she wishes to control the *transparency* of the erasing action.

The *arrow* tool is useful for selecting annotations in the edit window, either by directly clicking on them, or by engulfing them in a rectangle indicated — by clicking and dragging — with this tool. Once annotations are selected, they can be moved around on the presentation canvas. For fine positioning, the arrow keys, which nudge the selected annotations one pixel at a time, may be used.

The *lasso* tool is also a selection tool. It is different from the arrow tool in that it can be used to specify an irregular selection region. Clicking and dragging this tool will leave a trail which, when closed, will form the region in which annotations will be selected. This tool is useful when authors wish to quickly select multiple annotations arranged in non-rectangular regions.

The *image* tool introduces images in the background layer. Images may be selected from a file, from the clipboard (if available), or from images already available in any episode in the current lecture. Clicking and dragging with this tool specifies a rectangle. The associated image can then be clipped, zoomed or stretched to this rectangle. Alternatively, it may be displayed in its original size.

The *link* and *text* tools have not been implemented in version 1.0 of the recorder. In version 2.0 of the AudioGraph, they contribute to the process of creating links and text graphics.

The *scroll* tool allows the author to navigate through the presentation canvas when editing scrolls. In order to mimic the effect of the transparency scroll for overhead projectors, scrolling is only allowed vertically, even though horizontal scrolling could be supported by the system. The result of this navigation is recorded as a scrolling annotation.

The *sound* tool facilitates the editing of audio patches. Pressing this tool will start recording using the current sound input device and sound recording parameters such as the sample size, sampling rate, and compression method as specified in the application preferences panel. Pasting sounds stored in the clipboard directly into the presentation can also be used to create sound annotations.

Pressing the *option* key when clicking this tool opens a sound-editing dialog, as shown in Figure 4.4, which allows the author to record and replay sounds, and even type a textual transcript or comment associated with the sound annotation.

The *pause* tool introduces pauses in the presentation. Although internally pauses could be specified in milliseconds, the recorder offers a resolution of a tenth of a second.

The *colour* tool specifies which colour is to be used with the currently selected graphical tool. Colour is irrelevant for erasing annotations.

The *pen size* tool indicates how wide the graphical annotation lines should be. For the erasing tool, to ease its operation, the pen size value is automatically multiplied by a factor of four in order to cover a wider area.

The *previous episode* and *next episode* command buttons provide a means to navigate easily between episodes without having to switch to the lecture window.

The *undo* and *redo* command buttons provide a convenient alternative access to the unlimited undo functionality implemented by the recorder. When using a pen-based input device, this arrangement is very convenient.

3.5.2.2 Edit Console

The *edit console*, pictured in Figure 3.9, captures the sequence of recorded annotations, facilitates the sequencing of annotations and provides simple control of the playback functions.

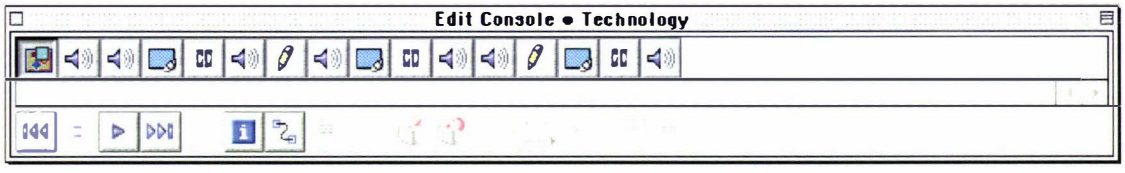


Figure 3.9. The edit console in version 1.0 of the AudioGraph recorder

Probably the most important innovation made when designing the new AudioGraph recorder in 1997 was to adopt what would later be called the *iconic paradigm* by Siglar in his Multimedia Authoring Systems FAQ [48]. Inspired by the sequential recording requirement carried forward from the early specification, we provided suggestive icons for each type of annotation available. Using this approach, the author can quickly spot areas of interest in the presentation, and can identify specific annotations with ease.

By clicking on an icon in the recording stream, the edit window reflects the selection as if playback would have occurred to that point in the presentation. Using the ability to select annotations in the recording stream, they can easily be rearranged in the episode by cutting, copying and pasting. Multiple selections are possible in the annotation stream using the usual

Macintosh keyboard modifiers, such as the *shift* key for multiple selections and the *command* key for non-contiguous selections.

We have also extended the iconic representation by using modifier mini-icons to indicate special attributes attached to annotations, as can be seen in Figure 3.10. In this fashion, it becomes very easy to distinguish between normal, erasing, highlighting and stealthy annotations at a glance.

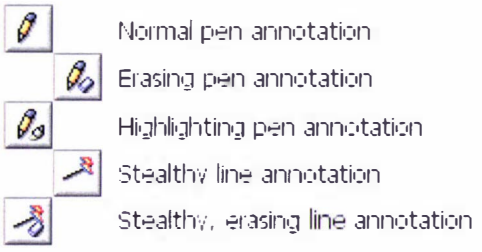


Figure 3.10. Annotations with special attributes

The Macintosh user interface uses a similar approach to visually emphasise the icons belonging to aliases and locked files. Similarly, the standard cursor is also adorned with various modifier mini-icons when a drag operation has different semantics due to modifier keys being pressed, such as alias creation, contextual menu invocation or copying of files rather than the standard file move semantics.

A textual description can be associated with most annotations. It was envisaged that it could be displayed in the edit console beside the icon, to provide a visual cue to the author. However, this functionality has not yet been implemented.

Since the PowerPoint [64] presentation file format is proprietary and we did not contact Microsoft to ask for a copy of the specification of the binary file format, we did not devise a direct import mechanism. Should we have done so, we would have had to adapt our application each time Microsoft updated the file format.

However, PowerPoint [64] can export a presentation as a Macintosh *Scrapbook* file containing all the slide images. The AudioGraph recorder can then import the *Scrapbook* file by transforming it into a lecture with episodes corresponding to each image. The images are included as the first effective annotations in each episode. This approach is very flexible, as *Scrapbook* files containing images ready for import can also be prepared manually or with the help of other tools as well.

In conversations with Paul Lyons at Massey University in 1998 [23], we explored various ideas of creating AudioGraph courses based on importing various documents. Although many lecturers use PowerPoint [64] presentations for their lectures, some lecturers prefer preparing their courses as text documents.

We then envisaged the possibility of importing a text document as the basis for an episode, and then using appropriate mechanisms in the AudioGraph recorder to partition the text into annotations. Using this approach, one might start with a much larger edit console resembling a text editor. Then, selecting sections of text, the author can start to partition them into components. Depending on the original text document, most sections of text could be very close to the transcript of sound annotations in the finished presentation.

Given the synthetic speech capabilities of the Macintosh platform, we then envisaged the possibility of automated sound production for the early stages of episode authoring. It is likely that until synthetic speech becomes substantially more realistic, it would not be acceptable on its own in educational material.

Although the AudioGraph recorder provides the ability to store transcripts with sound annotations, the sound synthesis capability has not been implemented yet.

Let us take a closer look at the controls in the edit console, shown in Figure 3.11.

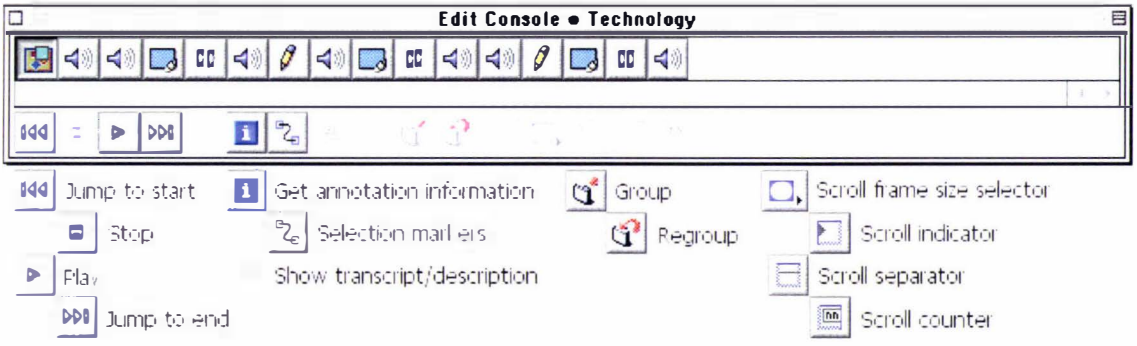


Figure 3.11. The controls in the edit console for version 1.0 of the AudioGraph recorder

The *jump to start* button positions the focus at the very beginning of the episode. Since the AudioGraph uses a single stream of annotations, the focus of the user interface passes through a single sequence of points, without any branching, merging or any other kind of parallelism.

The *stop* button stops the presentation once playback is underway. The *play* button starts playback from the current point of focus.

The *jump to end* button positions the focus at the very end of the episode. It is the last button in the group that provides playback and positioning services.

The *get annotation information* button opens up an editing dialog appropriate to the current selection in the stream. Using these editing dialogs, it is possible to alter annotation attributes and characteristics.

For instance, for pen, line, oval and rectangle annotations it is possible to alter the description, colour, pen size, and visual attributes. In addition, for arc annotations it is possible to alter the arc start and end values.

For image annotations, it is possible to select a new image source, alter the clipping or zooming attributes, and it is even possible to *vanish* the current image. This option introduces an erasing image annotation in the stream, which has the effect of hiding the corresponding image once it is encountered in the playback stream. To be valid, erasing image annotations must find themselves in the playback stream after the image they are intended to hide.

For a sound annotation, it is possible to edit the sound and its transcript, as seen in Figure 4.4. For pause annotations, it is possible to alter the pause durations.

The *selection markers* button toggles the display of the rectangular transparent markers that indicate which components are currently selected in the editing window.

The *show transcript/description* function has not been implemented yet, and it is the last function in the group related to annotation information.

The *group* button is enabled if multiple contiguous annotations are selected in the editing stream. In this case, its role is to replace the selected annotations with a single group annotation. When a single group annotation is selected in the editing stream, the role of this function reverses to replace the selected annotation with the contained group of annotations, becoming effectively an *ungroup* button.

The *regroup* button is available if the author has just disbanded a group annotation by making use of the *ungroup* functionality as described above. In these circumstances, regardless of the current selection, choosing the regroup function will absorb the previously grouped annotations once more into a group annotation. The regroup button is the last in the group that deals with grouping functionality.

The following controls are only enabled when editing scroll episodes. They control various markers that either indicate the current scroll frame, or can be used to determine the current position on the editing canvas.

The *scroll frame size selector* control allows the author to use a thick, thin or no scroll frame to indicate where the current focus is placed on the editing canvas.

The *scroll indicator* toggles small triangular makers in the left margin of the editing canvas that mark multiples of the scroll frame height.

The *scroll separator* toggles thin line markers positioned at boundaries in multiples of the scroll frame height.

The *scroll counter* toggles numbered markers in the right margin of the editing canvas that indicate the multiples of the scroll frame height.

3.5.2.3 Attributes Palette

The *attributes* palette window, shown in Figure 3.12 in its *basic* and *advanced* states, provides fine-grained control over annotation attributes.

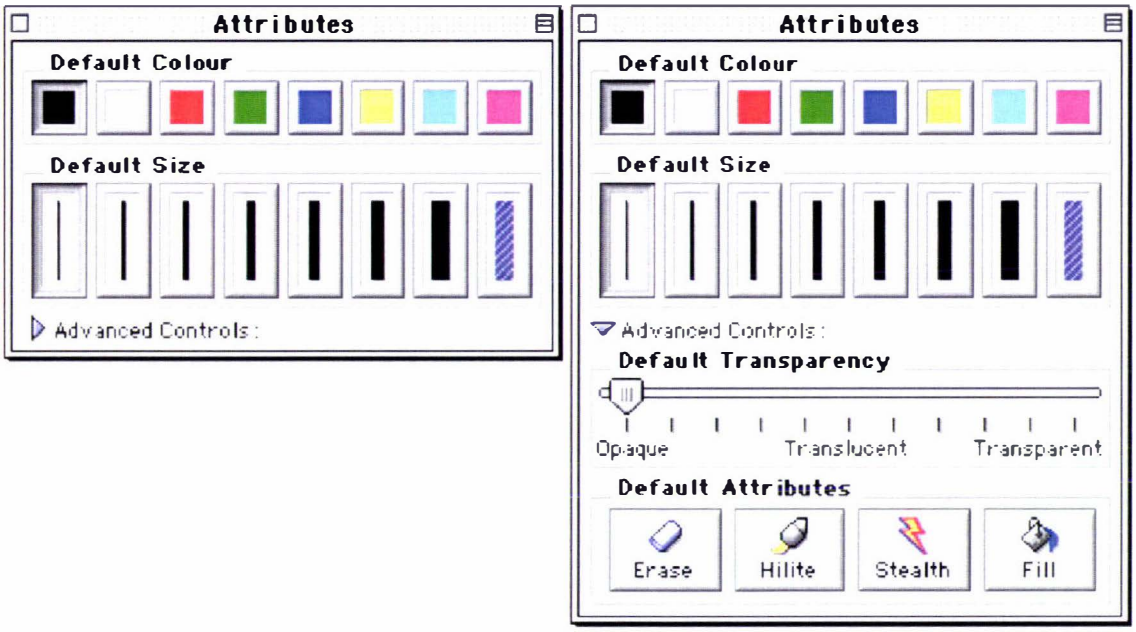


Figure 3.12. The attributes palette in version 1.0 of the AudioGraph recorder, in its two states

In the spirit of progressive disclosure, as also recommended by the Apple Human Interface Guidelines [158], we have chosen to provide access to more advanced controls only if authors specifically wish to use them. As such, the attributes palette is not displayed by default when opening an episode window, as the tools palette can also facilitate pen colour and size selection.

The attributes palette can be used to change colours and pen sizes quickly and easily, and it is particularly handy when using a pen input device, since tapping a button on the palette is much easier than having to make a selection from a menu.

The more advanced controls in the attributes palette enable the author to control the transparency of visual annotations and specify attributes to be associated with them.

The author is presented with a choice of eight default colours for the graphical elements in the episode. However, if this choice is insufficient for the user, by simply double-clicking on one of the colour controls, the author can personalise the colour palette. The standard Macintosh colour picker is presented, and authors may choose any colour in the full RGB space, which will be faithfully reproduced in the players as well, since 8-bit component values for the colour definition are propagated to the web-ready file format, allowing support for millions of colours.

The colour palette is persisted as a specific characteristic of the episode, and therefore the next time the episode window is opened, any custom colours are preserved in the palette.

3.5.3 Preferences

The AudioGraph recorder provides four preferences panels, controlling various aspects of the operation of the system — sound input, graphics, editing and general.

The preferences window contains three buttons that are shared among the various panels:

- *Factory Settings*, which restores the values of all controls on all the preference panels to the default recommended values.
- *Revert Panel*, which becomes available as soon as control values have changed on the current panel, and can be used to restore the panel settings to the values previously saved.
- *Save*, which saves the current state of the preferences, and which is enabled as soon as any control in any panel is modified.

In the following sections, we will examine each preference panel and the controls associated with them.

3.5.3.1 Sound In

The *Sound In* preferences panel, illustrated in Figure 3.13, controls the behaviour of the recorder while recording voice annotations. The *Recording Parameters* section controls the sample size, sampling rate, input source and compression used by default.

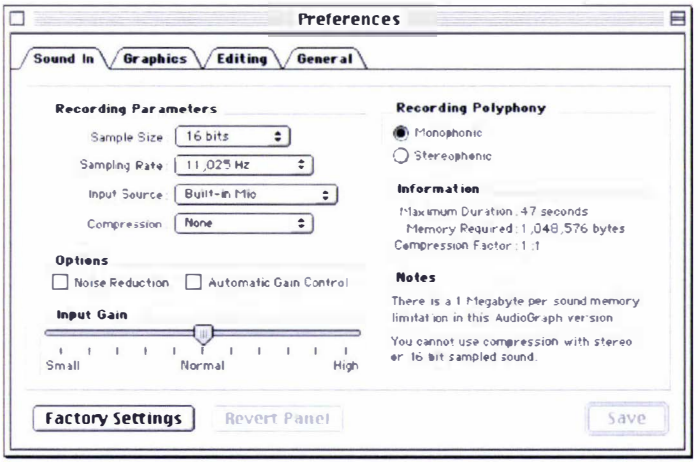


Figure 3.13. The Sound In preferences panel

Sample sizes may be 8 or 16 bits. Sampling rates may vary between 8 and 44kHz, with the exact values controlled by the specific capabilities of the hardware on which the application runs and the compression mode used. The hardware capabilities also dictate the input sources available. Beyond no compression, available formats were MACE [259] 3:1, MACE 6:1 — which are now obsolete [260] — and GSM 06.10 [227].

The GSM 06.10 [227] compression mode became available for storing voice annotations in the AudioGraph lecture file itself in version 2.0 of the recorder. Version 1.0 of the recorder only transformed the sound when exporting the lecture into the web-ready file format.

There are a couple of benefits associated with storing sound in GSM 06.10 format [227] in the lecture file. First, the lecture file sizes will be smaller, and load and save operations will be faster. Second, exporting lectures into the web-ready format is much faster, since sound annotations would no longer need to be converted to GSM 06.10 [227] during the export process, and could merely be copied across into the destination file.

However, the quality advantage of having uncompressed sound in the lecture document is sacrificed. Admittedly, if the target of the lecture authoring process remains web distribution using the AudioGraph web-browser plug-in, as it usually is the case at the moment, then this sacrifice may be well worth the benefits. On the other hand, when future versions of the AudioGraph may support alternate distribution formats — such as Flash [88], QuickTime [229] or MPEG-4 [249], as discussed in Section 7.3 — then trans-coding from GSM 06.10 to other modern high-quality formats such as MP3 [252], AAC [254] or Ogg Vorbis [242] is sure to yield much lower quality than direct compression from an uncompressed source.

The *Options* section controls whether *noise reduction* or *automatic gain control* dynamic processing should be used. The *noise reduction* option seeks to minimise hissing noises during pauses, and the *automatic gain control* option strives to adapt the microphone gain in order to keep the signal level close to the optimum, which is indicated by the boundary between the green and the red indicators in the sound recording dialog — see Figure 4.4 on page 103.

The separate input gain slider statically controls the input gain level. Authors may use this control to compensate for the particular characteristics of their microphone or the acoustics of the room where they are recording.

The *Polyphony* section controls whether the recorded sounds should contain mono or stereo samples. Some sound sources can provide stereo sound signals, and since advanced sound formats such as Ogg Vorbis [242] are envisaged as future enhancements for the AudioGraph system, this option is useful for providing a means to preserve high quality input signals.

The *Information* and *Notes* sections provide useful details about the sound capabilities and limitations derived from the current control settings. Version 1.0 of the recorder used a self-imposed limitation of one megabyte of memory per sound annotation, in an attempt to simplify processing and increase the robustness of the system. Use of very large sounds — beyond the one-megabyte limit — would have exposed us to the risk of running out of memory when running under systems earlier than MacOS X [160].

With this constraint in place, although uncompressed sounds with large sample sizes and sampling rates can only be recorded for very short durations, sound management is kept manageable and robust.

3.5.3.2 Graphics

The Graphics panel, shown in Figure 3.14, controls parameters that affect various authoring tools and the editing window in general. The *Pen* section dictates what default pen colour and size should be used for all new episode windows. The pen colour and size are recorded as characteristics of the episode itself, and therefore, if the user changes their values from the default, the next time the window is opened the author's preferred values take precedence.

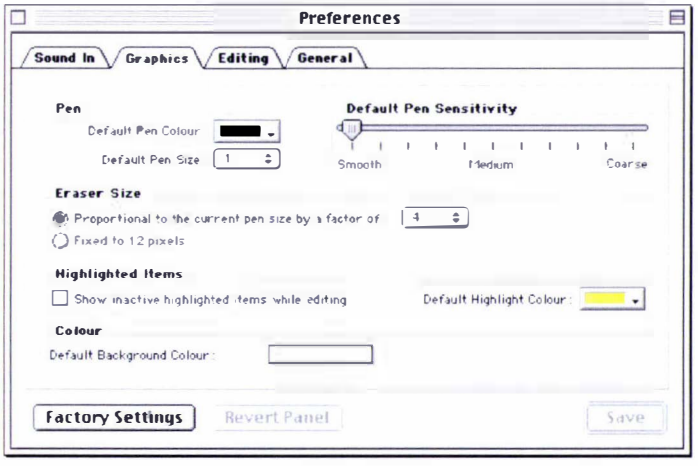


Figure 3.14. Graphics preferences panel

The *Default pen sensitivity* slider controls how frequently should samples of mouse position data be recorded when using the free-hand pen drawing-tool. The pen tool in the AudioGraph system does not use any form of interpolation in an attempt to smoothen free-hand drawings. All such drawings are poly-lines between the data points recorded while dragging the pointer in the edit window area.

The *Smooth* end of the scale means that all position information from all mouse events is recorded in pen annotations, generally leading to pen annotations with many sample points and in consequence smoother curves at the expense of storing more information. For the *Coarse* end of the scale, position information data is recorded more sporadically, leading to free-hand drawings with sharper corners, but more bandwidth-efficient.

The *Eraser Size* section facilitates the choice between a fixed and a proportional eraser-tool tip size. Users may adjust these values to suit their personal editing style.

The *Highlighted Items* section controls what default colour should be used by the highlight tool. The "Show inactive highlighted items while editing" option controls whether all highlighted

items should be visible at all times, or only the current highlight group. For a description of the behaviour of highlighted items and highlight groups, see Section 3.3.4.3.

The default background colour for all new episodes and scrolls can also be specified in this panel, again using the standard Macintosh colour picker.

3.5.3.3 Editing

The *Editing* panel, pictured in Figure 3.15, controls further settings related to slide and scroll editing. The default background size can be specified for all new slides. The default image-import options can be specified as well, with a choice of using the original sizes, clipping, stretching or zooming imported images.

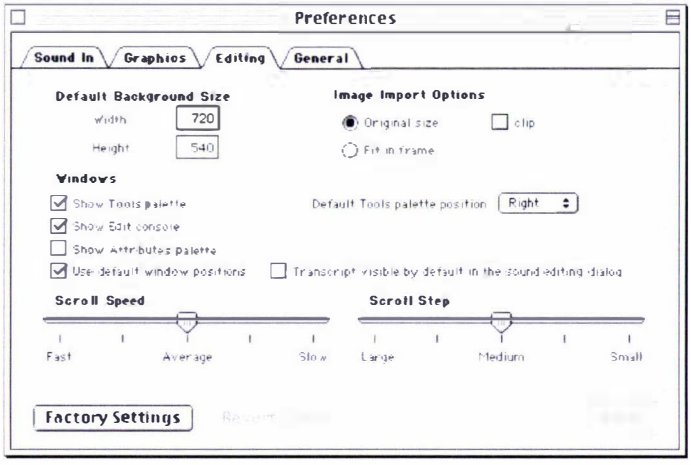


Figure 3.15. Editing preferences panel

The *Windows* options control the visibility and location of the floating palettes. To assist left-handed authors, the tools palette may be configured to pop up on the left of the edit window by default. The user may specify whether the default window positions should be used whenever a new episode window is opened. Alternatively, the latest window position before it is closed is saved and then used the next time the window is displayed again, allowing authors to maintain with ease their favourite window configuration.

The sound editing dialog box may display a transcript for the sound annotation, as shown in Figure 4.4. The preferences panel option indicates whether the transcript should be visible by default or not when the sound editing dialog box is opened.

The behaviour of the scrolling tool can also be affected, by specifying the scrolling speed and step. Authors may adjust the scrolling parameters to suit their working style or fine-tune their operation to the performance characteristics of their computer.

3.5.3.4 General

The last preferences panel, shown in Figure 3.16, may be used to control various characteristics of the application behaviour that do not fit readily in any of the other panels.

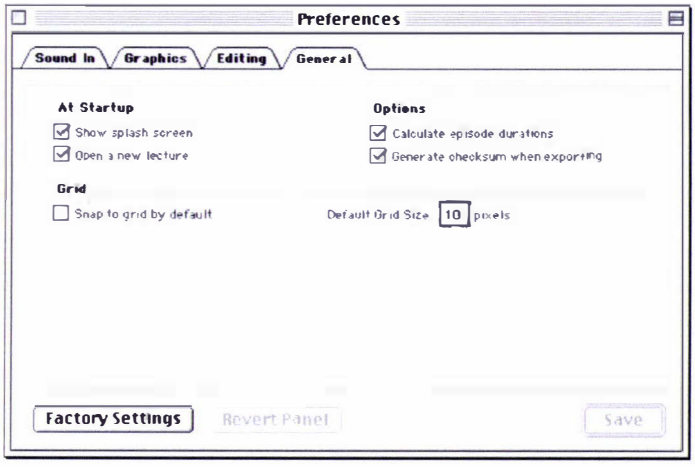


Figure 3.16. General preferences panel

In keeping with traditional behaviour of Macintosh applications, options are provided to display the splash screen or open a new lecture on application start-up.

The recorder may also be required to calculate episode durations or generate checksums when exporting files into the web-ready file format.

The episode window may use a grid to constrain all newly drawn or moved graphical annotations, and the grid size may be specified as well.

3.6 The AudioGraph Players

We have registered the **`application/vnd.audiograph`** MIME type [197] for the AudioGraph web-ready file format [198]. The recorder application uses a different, proprietary format for storing lecture-authoring data.

The AudioGraph recorder produces simple websites containing pages that require a web-browser plug-in capable of rendering streams that conform to the above MIME type to view the episode files exported in the web-ready file format.

Besides web-browser plug-ins, standalone applications can be developed to play AudioGraph episode files, similar to the Adobe Reader application [200].

For version 1.0 of the AudioGraph, we have developed a Macintosh web-browser plug-in, a Macintosh standalone player application and finished a Windows PC web-browser plug-in, based on an early, unfinished skeleton begun at Surrey University.

The Macintosh players share the same interface, seen here in Figure 3.17, providing an identical user experience for both the standalone player and the web-browser plug-ins. We chose to design the user interface so that it has a similar layout and it provides similar functions to interfaces used by other media player applications, such as the QuickTime plug-in player [228].

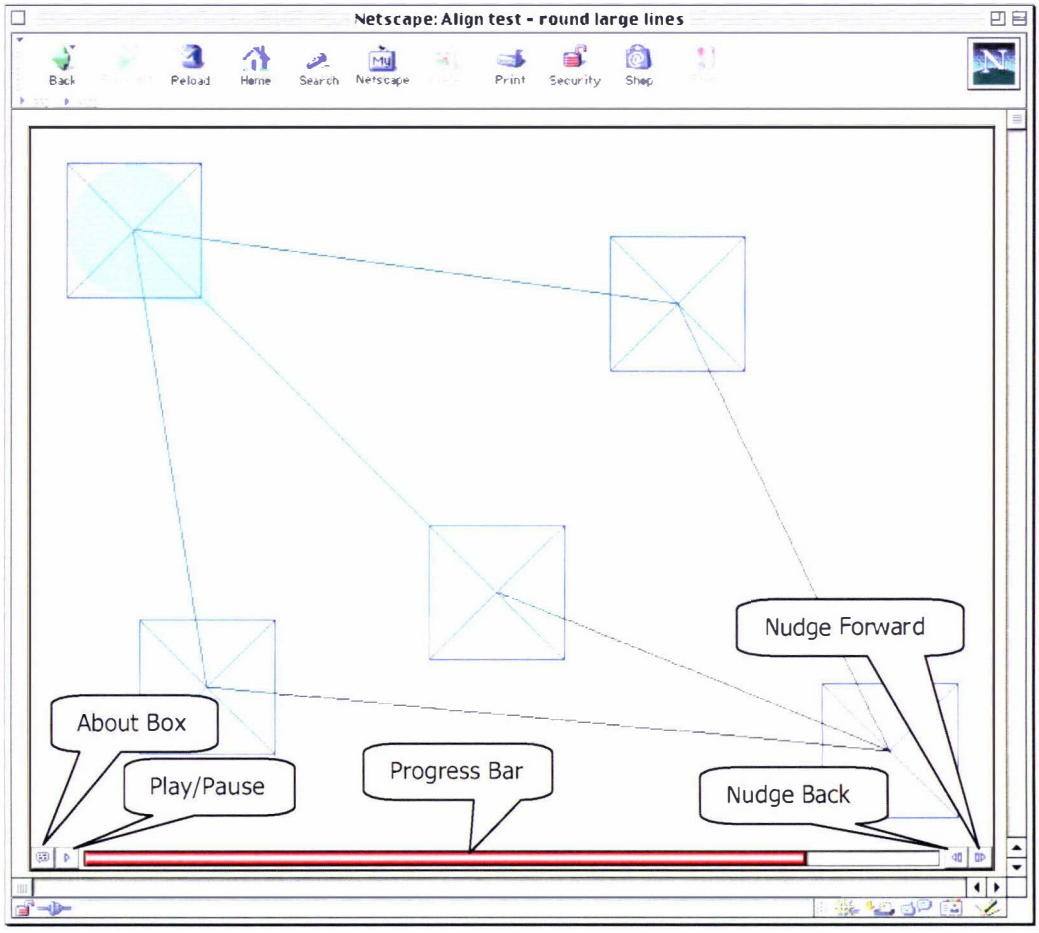


Figure 3.17. Browser plug-in example

The PC plug-in has a less advanced interface, providing only a scroll bar rather than a progress bar and associated fine-grained controls for controlling the presentation.

The browser plug-in controls — indicated by callouts in Figure 3.17 — are very simple to learn. By clicking anywhere in the plug-in display area the presentation can be started and stopped. In effect, the reach of the *Play/Pause* button (in the bottom left corner) is extended to the whole presentation area. The *progress bar* indicates the current position in the presentation. The colour of the bar also shows whether the presentation is running or it is paused; if it is green, the presentation is running, while the colour red means it is stopped.

The two arrow buttons on the right can be used to nudge the focus in the presentation by one annotation at a time. The 'text bubble' button in the left corner of the plug-in window can be used to display a splash screen — also called 'about box'.

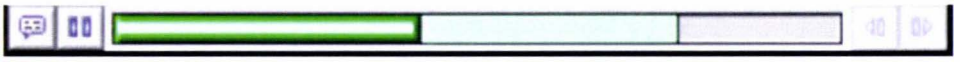


Figure 3.18. Load progress indicator in the browser plug-in progress bar.

A shadow indicator in the progress bar, shown in Figure 3.18, tracks the download progress. This feature is most useful for users with low bandwidth connections, giving good feedback about the amount of data available for playback. While the plug-in is waiting for more data to arrive, the playback stalls, and it resumes as soon as more annotations can be reconstructed from the newly arrived data.

Figure 3.19. Control features in the presence of linked presentations

For version 2.0 of the plug-in, when multiple episodes are hyper-linked in the same plug-in instance, new controls — as seen in Figure 3.19 — are available, and visual cues in the progress bar indicate the presence of multiple episodes.

Notice how the control bar contains a number of segments, each denoting an individual episode. The two new 'arrow' navigation buttons on the right can be used for jumping between episodes, just as the normal nudge buttons can be used for jumping between annotations.

Figure 3.20. Control features for small-size presentations

When the size of the presentation is too small to accommodate all controls comfortably, the plug-in hides all control buttons — as seen in Figure 3.20 — and presents just the control bar, which is sufficient to control all playback aspects in full, in conjunction with the display area, which plays the role of the expanded *Play/pause* button. This capability has been taken even further in version 2.0 of plug-in, since episodes may request that no controls be displayed at all, leading to a user experience similar to the playback of Flash [87] animations.

3.7 Summary

In this chapter, we have introduced the AudioGraph web-based lecturing system. To begin with, we sought to help readers imagine the system's intended use in lecturing. To this end, we explained the AudioGraph system's structure and briefly outlined a typical usage example, which is elaborated upon in Appendix A.

We then developed our main requirements, guided by our research objectives, our analysis of the AudioGraph prototype [1] and informed by our assessments presented in Chapter 2. We presented the main AudioGraph concepts that addressed our requirements, the web-ready file

format and the authoring, playback and display models. We have then shown the evolution of the AudioGraph system components over time and clarified the contributions of the author in this context.

Subsequently, we have presented the main user interface elements of the AudioGraph recorder and player applications, and reviewed their functionality and capabilities.

In the next chapter, we will examine the software engineering context in which our research has been conducted, which has lead to the effective realisation of the system.

4 Software Engineering Context

In this chapter we will explore the software engineering context of our research. As human interface design principles and guidelines have a crucial influence over system design, we will explore their implications for our work.

Due to our overriding bandwidth economy goal, data compression — for both images and sound — plays a vital role in our efforts, as cost-effective web delivery relies heavily upon it. We will briefly describe the challenges of image and speech compression and introduce the technologies that best fit our requirements, PNG [218] and GSM 06.10 [227].

From a software engineering perspective, the advent of *design patterns* [127] has had a profound impact on how designers of object-oriented systems describe and communicate their design decisions. We provide a brief introduction to this subject and discuss the main design patterns that can be discerned in the AudioGraph system design.

Henessy and Patterson recommend, as the first quantitative principle of computer design, to *make the common case fast* or, more precisely, *make the frequent cases fast, and the rare case correct* ([141] pp. 39-40, 126). In the pursuit of quality and performance, we have found that the spirit of this excellent advice could be observed at work in many different contexts, and have highlighted this in the relevant sections.

4.1 Human Interface Design

Good human interface design is of vital importance when developing any kind of system in order to present its users with the best possible experience. Jef Raskin, the creator of the Apple Macintosh user interface, suggests that "If a system's one-on-one interaction with its human user is not pleasant and facile, the resulting deficiency will poison the performance of the entire system, however fine that system might be in its other respects" ([128] pp. *xix*).

On the subject of the role of interface design in the software development cycle, Raskin argues that the user interface design should be considered integral to the requirements analysis phase. He recommends that once the system's task is known, first of all the interface should be designed, and then the implementation should realise that interface design. To users, the interface represents the system. As long as the interface satisfies the users, the underlying technical details of the system are generally irrelevant to them ([128] pp.5).

Although his book was published well after the core of the author's software design and development effort for this work was complete, we have independently followed such an approach, as described in Section 1.5.

Cooper also champions *interaction* design ([131] pp. 23), in the same spirit as Raskin. He sees *interaction* design as a synergy of *conceptual* and *behavioural* design. While *conceptual* design seeks to gain good knowledge of the target user base and find what is most valuable to them in the first place, *behavioural* design aims to establish how the software elements should act and communicate in this context, in a goal-driven, *outside in* design strategy.

He contrasts this to an apparently industry prevalent *inside out* design strategy, predominantly driven by the technical structure of a system, which is the cause of much interface clumsiness[1].

In an attempt to alleviate such problems and help designers to create effective interfaces, a considerable number of principles and guidelines for interaction design have been developed. For example, Mayhew [129] and Smith and Mosier [130] have collected comprehensive platform-independent guidelines.

Various industrial companies and consortia have also issued style guides that relate to particular platforms or software toolkits, and are therefore usually more concrete and specific than the platform-independent guidelines ([135] pp. 1534).

The most relevant international standard on human-computer interaction is ISO 9241 [137], which contains 17 parts and two amendments, at various stages in the standardisation process.

Part 10 of the standard, which is a final, published International Standard and a European Norm, describes the general high-level principles of ergonomic dialog design, and suggests interfaces should have the following properties:

- Suitability for the task.
- Self-descriptiveness.
- Controllability.
- Conformity with user expectations.
- Error tolerance.
- Suitability for learning.
- Suitability for individualisation.

Although these general principles contribute to defining commonly agreed goals of user-centred design, they require a substantial amount of interpretation in order to prove useful and are hardly applicable without additional descriptions and examples.

[1] An interesting parallel can be drawn here with the evolution of microprocessor assembly languages. In the early days, hardware engineers designed microprocessor assembly languages — which can be considered to be the microprocessor's user interface — without much consideration for their user base, the software engineers that had to write efficient firmware or compilers. It was two decades later that Hennessy and Patterson introduced their quantitative approach to computer architecture design, which led to the creation of orthogonal, more efficient Reduced Instruction Set Computer chips by optimising instruction set designs based on measurements of software usage patterns [141].

On the other hand, the user interface design guidelines recommended by Apple Computer [158] embrace these principles and refine them in the context of the Macintosh user interface, encouraging consistency for the applications developed for this platform.

As the Macintosh has been our primary target platform, it seemed most appropriate, keeping in mind the spirit of the wider principles outlined above, to seek to weave our interface designs in accordance with the Apple Macintosh Human Interface Guidelines [158]. We will review these guidelines in the following sections, and show the implications they have had for our designs.

4.1.1 Human Interface Design Principles

A set of useful principles has been accumulated in the process of designing human interfaces for computer systems. By making use of a defined set of design principles, software developers can build products that meet the standards of the platform they are targeting.

It is a reasonably safe assumption that in general not all guidelines can be satisfied simultaneously in all circumstances. Under such circumstances, the various trade-offs required by the competing guidelines are weighed, and a decision must be made based on the most valuable features that are being sought.

The *make the frequent cases fast, and the rare case correct* principle ([141] pp. 126) can also be applied in various guises to human interface design. For instance, if a particular user action is expected or observed to be frequent, then simplifying its operation — e.g. by reducing the number of commands or clicks required, providing shortcuts or by minimising the distances to be travelled with the mouse — can improve productivity and enhance the user experience.

4.1.1.1 Metaphors

We can take advantage of people's knowledge of the world around them by using metaphors to convey concepts and features in our applications. Generally, metaphors involving concrete, familiar ideas should be used, and they should be kept as simple as possible, so that the largest possible proportion of users can have a familiar set of expectations to apply to computer environments.

Although metaphors in the computer interface may suggest certain uses for various components, that use does not necessarily have to constrain the implementation of the metaphor.

To achieve the best effect, a balance must be sought between the metaphor's suggested use and the ability of the computer to support and extend the metaphor.

For instance, the AudioGraph recorder edit window may be said to reflect the canvas metaphor or the scroll metaphor, depending on whether one is editing a slide or a scroll. Familiar actions available to users of a scrolling projector are available for authors working with scrolls.

Figure 4.1. Locking a lecture prevents unwanted changes

Another example of a metaphor at work is the small 'lock' icon in the lecture window, as seen in Figure 4.1. When clicked, the lock toggles with a brief animation between the two states, and can be used to protect the lecture from unwanted changes by 'locking' it.

4.1.1.2 Direct Manipulation

Direct manipulation allows users to feel that they are directly controlling the objects represented by the computer. According to the principle of direct manipulation, an object on the screen remains visible while a user performs physical actions on the object.

When the user performs operations on the object, the impact of those operations on the object is immediately visible. In addition to expecting physical results from their actions, users want their tools to provide instant feedback.

For instance, in the AudioGraph recorder, visual annotations are apparent as they are drawn. A matching icon appears in the edit list as the annotation has been recorded. If a set of annotations is selected, changing a certain attribute by using the attributes palette such as colour, transparency or pen width applies immediately to all selected items. In the players, as the progress bar is dragged, the current annotation changes accordingly, allowing users to navigate effortlessly to any desired section of the presentation. There are countless other such instances where the principle of direct manipulation is applied throughout the various components of the AudioGraph system.

Users generally want to see what actions are available at any given moment. For this reason, the *Edit* menu commands are updated accordingly, depending on the type of component currently selected, the type of component available in the Clipboard and the last action performed.

These are all elements form the current execution context, and they can be used to enrich the user experience, freeing the user from having to remember what each specific command is likely to affect. This capability is especially useful for commands that operate on objects that are not directly within the user focus, such as the Clipboard contents or the top of the undo stack.

For example, as shown in Figure 4.2, the context of the current selection is a pen annotation, the Clipboard contains a rectangle annotation, and the last action performed by the author was drawing an oval annotation.

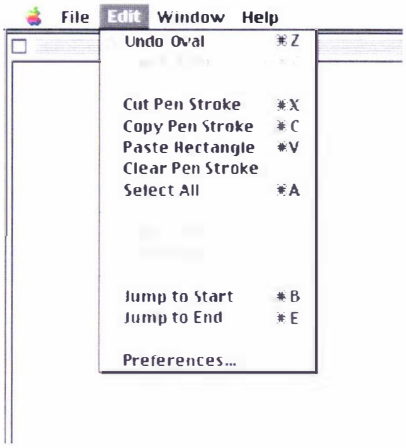


Figure 4.2. The commands in the Edit menu reflect the current execution context

Users should be warned if certain actions are likely to have severe repercussions — before any damage is done and while there is still a chance to avoid losing data. As a result, potentially risky operations are protected by alert dialogs that require users to confirm their choice.

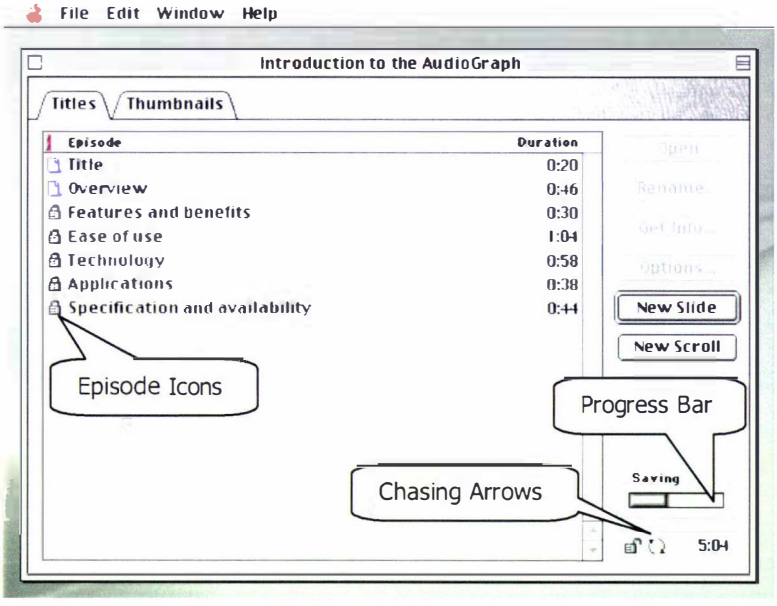


Figure 4.3. Various forms of feedback are available when saving a lecture

Animation, when used judiciously, is one of the best ways to convey that the most recently invoked action is being carried out. For example, in Figure 4.3 the chasing arrows in the lecture window are meant to reassure the user, during the potentially lengthy process of saving a large AudioGraph lecture to disk, that the task is progressing without problems.

A small progress bar appears as well, visually indicating the evolution of the saving process. You may also notice that the episode icons gradually change from locks to normal slide icons as they are saved, indicating which episodes have already been saved to disk and are ready for editing once more.

4.1.1.3 See-and-Point

Users interact directly with the screen, selecting objects and performing activities by using a pointing device such as a mouse or a pen tablet, to point, click and drag elements in the application's user interface.

The Macintosh desktop itself works according to two fundamental paradigms that share two basic assumptions:

- Users can see on the screen what they are doing.
- Users can point, click and drag objects on the screen.

Both paradigms are based on the *noun-then-verb* mode of user interaction.

In one paradigm, the user selects an object of interest — the noun — and then chooses the actions to be performed on the object — the verb. All actions available for the selected object are listed in the menus, and the AudioGraph menu items even adapt according to the current execution context as seen in Figure 4.2, so that users who might be unsure of what they could do next can refresh their memory by scanning through the menus. Actions that are not currently available are suitably dimmed, to indicate they are inactive. Using this approach, users can choose among the available actions at any time without having to remember any particular command or name.

In the second paradigm, the user may drag an object — the noun — onto some other object that has an action — the verb — associated with it. For instance, as the graphical representations of annotations are dragged around in the edit window, they move with the cursor. Although the user does not have to choose an action from the menus, it is perfectly clear what happens to the object as it is dragged.

4.1.1.4 Consistency

By designing the interface with consistency in mind, we can help users to transfer their knowledge and skills from one application to another. In this manner, by predominantly using the standard elements of the Macintosh interface in similar ways we may achieve consistency within our applications and benefit from consistency across other applications as well.

To be effective, applications should seek to be consistent in a number of different ways. Consistency in the visual interface helps people learn and then easily recognise the graphic language of the interface — for example, once users know what a checkbox looks like, they do not have to learn another symbol for making binary choices. Consistency in the behaviour of the interface means that people have to learn just once how to do things such as clicking and dragging, and then they can explore new applications or new types of features using skills that

they have already mastered. In general, consistency benefits the typical user, who usually divides working time among several applications, and it benefits the software creators as well since their users can build on prior experiences with elements in other applications when learning how to use a new application.

Not surprisingly, the most difficult kind of consistency to achieve is matching everyone's expectations. Because we often face a wide audience, with a broad range of expertise, it is difficult to succeed in meeting all expectations in full. We can mitigate this problem by carefully weighing the consistency issues in the context of the primary target audiences — the average lecturer and student, in our case — and by identifying as accurately as possible their primary skills and needs. The general simplicity of our interfaces reflects this guideline.

4.1.1.5 WYSIWYG (What You See Is What You Get)

Application features should not be obscured by abstract commands. People should be able to see what they need, when they need it. For example, menus present lists of commands so that people can see their choices instead of having to memorise and type command names. People should be able to find all the available features in application without any guesswork involved.

Sometimes, however, it is beneficial to "hide" advanced features initially in order not to overwhelm novice users. When this is the case, it must be done in such a way as to give users clues about where and how the extra choices can be found. A stepped interface, that reveals relevant information to users in steps, shows the choice most users want most of the time while providing a way for the user to get more choices as they desire.

For instance, the default windows presented to the user when opening a slide for the first time are the edit window, the tools palette and the edit console. The attributes window is not visible by default, since it contains controls that are more advanced. The attributes palette itself is further divided in two sections, with the even more advanced controls for transparency and special annotation attributes protected by another step, as seen in Figure 3.12. This is yet another example of the *make the frequent cases fast and the rare case correct* ([141] pp. 126) principle at work.

In addition, we have ensured that there is no significant difference between what the author sees on the screen when editing an episode in the recorder and what the student views in a player. Authors are in charge of both the content and the structure of the lecture, both visual and temporal. When the user makes changes to any episode, the results can be viewed and tested immediately, without requiring the user to export the material into the player format or make any mental calculations to determine how the currently visible presentation will look when viewed by a student.

4.1.1.6 User Control

Since humans learn best when they are actively engaged, they should consistently be the ones to initiate and control actions, not the computer. Situations where the computer acts and users merely respond, by choosing from a limited set of options, should be avoided. The temptation of having the computer chaperone the user, offering only those alternatives that are judged "good" for the user or that "protect" the user from having to make detailed decisions should be eschewed as well. Such approaches mistakenly put the computer, not the user, in control.

The key to successful interface design is to create a balance between providing users with the capabilities they need to get their work done and preventing them from ruining their own work and damaging their data. To assist in this quest, the AudioGraph recorder implements *multiple undo* functionality, in practice only limited by the memory and storage capabilities of the computer. As a result, even if users perform rash actions, they may have a chance to repair the damage done through unwanted changes by backing out through the undo stack until they are satisfied with the state of the episode they are editing.

In cases where a user may destroy data accidentally, such as when closing a lecture document without saving it, we help the user by providing a warning, as an alert box, to notify the author of the potentially undesirable situation and still allow them to proceed, if they confirm that this is indeed what they want. This approach "protects" authors while keeping them in control.

4.1.1.7 Feedback and Dialog

Users must be informed about what is happening within their applications at all times. It is imperative to provide feedback as users perform tasks and to make that feedback as immediate and relevant as possible.

For instance, when a user initiates an action, such as dragging the cursor to create an annotation, we provide immediate feedback that the application has received the author's input and is operating on it by drawing the annotation as appropriate at the time. For long operations, such as loading and saving lectures, we provide as much information as possible about how long the operations are likely to take by displaying a special progress bar, as seen in Figure 4.3.

We must also strive to provide direct, simple feedback that users can understand. Most people would have no idea what a message like "An exception has occurred. Error code -108." could possibly mean. It would be considerably more helpful if the message spelled out exactly which situation caused the error — in our example, not enough memory was available for the computer to complete the current task — so that the user could perhaps understand how to avoid the situation in the future.

As an example of this principle at work in the AudioGraph context, in situations where memory is scarce, the AudioGraph players attempt to degrade their performance gracefully by gradually reducing the pixel depth in use, and also providing various messages explaining what is going on and suggesting potential remedies.

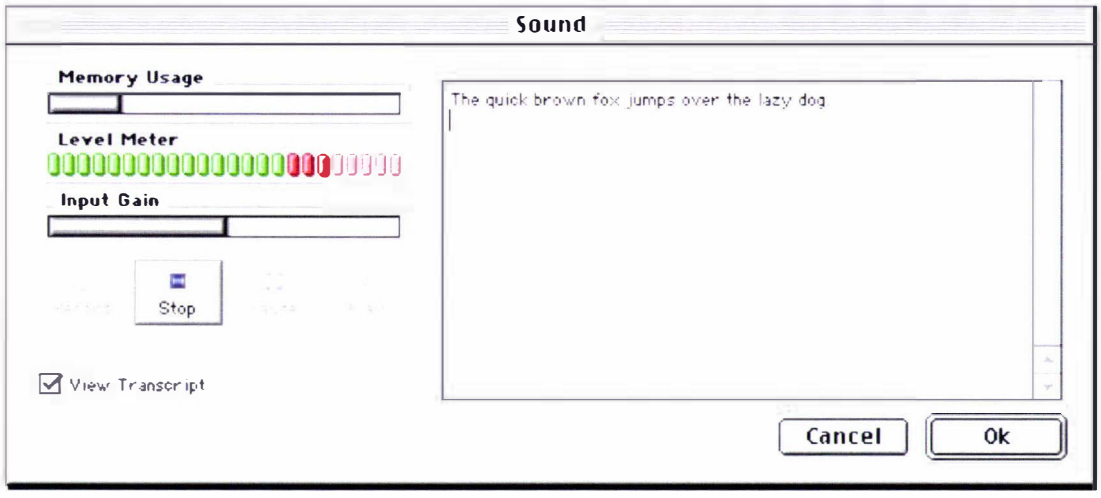


Figure 4.4. Dynamic feedback during sound recording

Similarly, when recording sound, the current microphone level is made clearly visible, as shown in Figure 4.4, using a familiar interface simulating a bank of green and red LED indicators. Progress bars indicate the amount of memory available and the current input gain level.

4.1.1.8 Forgiveness

Users may also be encouraged to explore an application by building in forgiveness. Forgiveness means that actions performed on the computer are generally reversible. People need to feel that they can try things without damaging the system. It is essential to create safety nets for people so that they feel comfortable learning and using new applications.

For example, the AudioGraph recorder provides a multiple-undo capability that is effectively limited only by the amount of memory and disk resources available. If unwanted changes are made to a lecture, it can be easily restored back to the state where it was last saved by using the *Revert* command. The lecture window also has a little lock icon, as seen in Figure 4.1 and Figure 4.3, that can be used to protect the lecture from unwanted changes by 'locking' it.

In the same spirit, it is important to warn users when they initiate a task that could cause irretrievable data loss. Alert boxes are a good way to warn users of this kind of situation. However, when options are presented clearly and feedback is appropriate and timely, using a program should be relatively error-free and in consequence alerts should be a rare occurrence.

4.1.1.9 Perceived Stability

Since computers often introduce a significant new level of complexity in people's daily experience, they need some stable reference points to assist them in developing coping strategies. The Macintosh interface is designed to provide a computer environment that is understandable, familiar and predictable.

To give users a visual sense of stability, the Macintosh interface provides the desktop, a two-dimensional space on which objects are placed. It also defines a number of consistent graphics elements — menu bar, window border, and so on — to maintain this illusion of stability. Note that it is the perception of stability that should be preserved and not stability in any strict physical sense.

For our part, we aim to give users a conceptual sense of stability by having the AudioGraph interface provide a clear, finite set of objects and a clear, finite set of actions to perform on those objects. Even when particular actions are unavailable, they are not eliminated from a display but are merely dimmed and updated accordingly. For instance, when there is no action to redo, the command is dimmed and displays "Can't Redo", as seen in Figure 4.2.

4.1.1.10 Aesthetic Integrity

Aesthetic integrity means that information is well-organised and consistent with principles of visual design. This means that things look good on the screen and that the display technology is of high quality. Since people spend a lot of their time working while looking at the computer screen, it is important to design user interface to be pleasant to look at for a long time.

Graphics — icons, windows, dialog boxes, and so on — are the basis of effective human-computer interaction and must be designed with this principle in mind. Therefore, cluttering the screen with too many windows, confusing the user with arcane icons, or crowding dialog boxes with large numbers of controls should be avoided as much as possible.

In general, graphic elements should be matched with users' expectations of their behaviour. For instance, push buttons appear as though they push in rather than slide sideways. Indicators in sliders slide along to change values. These behaviours map to people's expectations of how these elements should behave.

We have spent considerable effort in pursuit of this principle, developing meaningful and attractive icons and structuring the user interface in aesthetic patterns, as can be seen in the examples throughout this chapter and also in Chapter 2. In order to enhance the usability of the interface, as an overriding goal, we have kept the graphics simple and suggestive. For instance, the lock icon, as seen in Figure 4.1, even though not a part of the standard Macintosh interface, conveys very clearly and simply the meaning of the feature it controls.

We have made sure to follow the graphic language of the standard user interface and have cautiously avoided changing the meaning of any standard items. For example, if we were to use checkboxes sometimes for multiple choices and at other times for exclusive choices, we would have diluted the meaning of the checkbox user-interface element.

At times, it may prove beneficial to give users some control over the appearance and layout of their computer environment. This allows them to display their own style and individuality. It also reduces the designer's burden of trying to create an interface that appeals to every user. When a user sets up her computer environment in a certain layout, it should stay that way until the user changes it.

As a manifestation of this guideline, the AudioGraph recorder can remember the window positions for each lecture, episode and editing palettes, so that the next editing session can begin with exactly the same layout, which is presumably most comfortable to the author.

4.1.1.11 Modeless operation

Most features in an application should be modeless, allowing users to do whatever they want at any moment they wish. Using specific editing modes — although relatively common in earlier computer systems — should be avoided if possible because a mode typically restricts the operations that the user might perform while it is in effect.

Modal operation locks the user into one operation and does not allow the user to work on anything else until that operation is completed. In contrast, modeless operation allows the user to initiate various independent operations at the same time and thus gives the user more control over what he or she can do on the computer and in an application. As much as possible, we want to preserve the user's ability to be in control of the task and the order of operations.

This is not to say that modes should never be used in applications. Sometimes using a mode is the best way of addressing a particular problem. Most acceptable modes fall into one of the following categories:

- They are long-term modes, such as word processing as opposed to graphics editing. Seen in this light, each application can be considered a mode in its own right.
- They are short-term, "spring-loaded" modes, in which the user must constantly do something to maintain the mode. Examples are holding down the mouse button to scroll text or holding down the Shift key to extend a text selection.
- They are exceptional, alert modes, in which the user must rectify an unusual situation before proceeding. Such modes should be kept to a minimum.

Other modes are acceptable if they do one of the following:

- They emulate a familiar real-life situation that is itself modal. For example, choosing different tools in the tools palette — shown in Figure 3.8 — resembles the real-life choice of physical drawing tools. Similarly, scrolling the transparency roll on an overhead projector is modal, since the presentation does not continue until the scrolling has ceased. Concordantly, recording an AudioGraph scrolling action is modal.
- They change only object attributes, not behaviour. Specifying the default colour, transparency, pen size or special attributes for annotations are all valid examples.
- They block most other normal operations of the system to emphasise the modality, as in error conditions incurable by the software — for example, a dialog box might disable all menu items except Close.

If an application uses modes, there must be a clear visual indicator of the current mode, and the indicator should be near the object most affected by the mode. For instance, depending on which tool is chosen in the tools palette — depicted in Figure 3.8 — the pointer may look like a pen, and arrow, a lasso, and so on. It is very easy for users to get into or out of such editing modes, by simply clicking a different tool.

4.1.1.12 Knowledge of the Target Audience

Probably the most important first steps when designing software are identifying and then taking the time to understand the target audience. In the case of the AudioGraph system, the primary target audience consists of lecturers and teachers preparing course material, and their students.

As a result, it has been vital to ensure that the graphics and concepts presented in the interfaces of the AudioGraph system were well known by our target audience. For instance, the drawing tools, as shown in Figure 3.8, look and behave as they do in virtually all graphics applications on the Macintosh. Controls for the player applications, as shown in Figure 3.17, Figure 3.18 and Figure 3.19, are organised in familiar patterns, providing similar functions to those in other player applications, such as the QuickTime plug-in player [228].

In an attempt to anticipate what authors would need to accomplish, we have provided an easy way to import presentations generated in Microsoft PowerPoint [64] or any other arbitrary sets of images as a starting point for creating course material with AudioGraphic annotations. In discussions with colleagues [23], we have also envisaged a means of extending the iconic edit list functionality to provide for the creation of AudioGraph presentations from text documents, and perhaps provide text-to-speech capabilities as well.

4.1.2 Managing Complexity

The best approach to developing software that is easy to use is to keep the design as simple as possible. The general design guideline that simple design is good design applies directly to the discipline of human-computer interactions.

As a result, striving to simplify the interface should be a constant endeavour in the face of pressure for new features, since a simple interface increases the likelihood of meeting or exceeding user expectations. Complex interfaces have much higher chances of alienating most users, by frustrating their efforts and abusing their patience.

4.1.2.1 Using Progressive Disclosure

Progressive disclosure is one way to reduce the complexity of a design. It allows us to present the most common choices to users while initially hiding additional information or choices that are more complex. Progressive disclosure helps us develop the interface so that it is easy for novice users to learn and includes the features and power that advanced users may desire.

For example, in version 1.0 AudioGraph recorder, the advanced controls for transparency and special attributes in the attributes palette were initially hidden, as seen in Figure 3.12. A progressive disclosure triangle controlled their visibility by expanding or collapsing that section of the palette.

4.1.2.2 Implementing Preferences

Preference settings are user-defined parameters that the software remembers from session to session. Preferences can be a way for the application to offer choices to users about how the application runs.

Preferences often affect the behaviour of the application — such as the input source used for sound recording — or attributes of the content created with the application, such as the default episode background colour.

A preference should be a setting that the user changes infrequently. We must avoid providing choices that are likely to be frequent within one session through the preferences mechanism. Menu items, palettes and other interface elements closer to the usual focus of the user are better suited for such features. This is another instance where the *make the frequent cases fast and the rare case correct* ([141] pp. 126) principle proves itself valuable yet again.

By choosing the right way to implement a feature, we can give users the flexibility to choose, in their own way, their preferred method of working.

The AudioGraph recorder preferences panels are described in Section 3.5.3.

4.1.3 Future Design Decisions

For design decisions regarding features in the application, we always need to weigh their costs against the potential benefits. Every time we would like to add a feature to the application, we should consider the following factors:

- The application will get larger.
- The application may become slower.
- The application's human interface is likely to get more complex.
- More time is spent in development rather than refinement of existing features.
- The application becomes more difficult to document.
- The number of possible user errors increases.
- Every new feature may have an (adverse) impact on existing features.

This section presents several additional factors that should to be taken into consideration when adding features to the AudioGraph suite of applications.

4.1.3.1 Features Inspired by Market Pressures

It is most important to think about the users of the software and keep their interests foremost when making new design decisions.

Market pressures can sometimes cause developers to try to implement features that they feel are necessary, even when they might not have the resources to develop those features fully. Because a good review in the press can make an application successful, developers are sensitive to meeting the expectations of reviewers as well as their target audiences. Many times reviews include information on whether or not applications have certain features and the right number of new features. The pressure of competition is intense, and developers may often feel that they must make decisions based on market pressures[2].

4.1.3.2 Feature Cascade

When deciding whether or not to add features to a system, we must think about whether the benefits to users of adding further capabilities outweigh the additional development efforts, growth in size, and reduction in running speed that the features might cost.

---

[2] For instance, SMIL [187] support in the AudioGraph recorder might be an example of such a feature suggested by market trends. Since the AudioGraph project has dedicated itself to providing a low-bandwidth solution, the verbosity of SMIL [187] — due to its XML [169] foundations — goes against it.

However, since SMIL [187] is now emerging as a standard in multimedia authoring and it is already largely supported by QuickTime [229], it is likely that at some point in the future the AudioGraph recorder will be adapted to export SMIL [187] presentations as well. Fortunately, the *builder* pattern [127] (pp. 97) adopted in the recorder export section and the regular structure of SMIL [187] make it comparatively easy to provide such support.

When developing a relatively simple application, it may be very tempting to include additional features that users claim they want. It takes a lot of restraint to stick to the original intent of the application. We must be wary of the feature cascade, because it can often reduce the overall effectiveness of the application and add unwanted complexity to it.

4.1.3.3 The 80 Percent Solution

During the design process, we may discover problems with the software design. The 80 percent solution may help to determine how to solve those problems. This approach suggests that the design should be considered acceptable if it meets the needs of at least 80 percent of the users.

If we were to try to design solely with the power users in mind — which typically account for 20 percent or so of the target users — the majority of the users would most likely consider the design unwieldy. Even though those 20 percent most probably have excellent ideas and probably think a lot like the developers themselves, given the current culture, the majority of the user base most likely will not have similar tastes and habits to the 20 percent who are elite users of the system.

To find the 80 percent solution, it is important to involve a broad range of users in the design process. We were fortunate to be able to trial our suite of applications at Massey University, and have gained a wealth of information through the feedback we have received. We are therefore confident that the AudioGraph applications may serve our users well, and look forward to continued cooperation while the AudioGraph system continues to develop.

4.2 Data Compression

In 1948, Claude Shannon wrote an outstanding paper — *A Mathematical Theory of Communication* [140] — that was to set the foundation for modern information theory. His definition of entropy of symbols in communication channels, and the associated encoding scheme paved the way for modern loss-less compression algorithms, as used by image compression formats such as PNG [220] and JPEG 2000 [215].

The key to communication consists in conveying patterns through transmission channels. The goal of communication is to enable the recipient to observe the patterns coming from the sender.

Messages are typically built from a finite number of symbols. Therefore, we can associate probabilities with the various symbols based on the frequency of apparition in the transmission channel. From this foundation, we can build an encoding scheme that associates fewer bits for the high-frequency symbols and more and more bits for the less and less frequent symbols. As a result, the absolute minimum of information could be transmitted, provided the sender and

recipient are able to share the same encoding, and therefore the original message symbols could be reconstructed at the destination.

In a way, this can be seen as a manifestation of the *make the frequent cases fast and the rare case correct* principle ([141] pp. 126), given that the more frequent symbols require fewer bits for representation and are therefore faster to transmit, assuming constant bit transmission time, which is typically the case.

Due to the applications in telephony — for the transition from analogue to digital signal transmission — that were being investigated at the time, Shannon was ultimately concerned with noisy channels. However, he also investigated the noise-free channels as a special case of interest. Given the relatively low error rates in modern digital transmission systems, we are mainly concerned with the noise-free case as well.

4.2.1 Image Compression

When compressing image data, we are faced with two major choices.

If we wish to loose no information whatsoever, and desire to reconstruct the image data perfectly, then we must compress the stream of bytes that makes up the image. In this approach, we do not take advantage of the fact that the data source we wish to compress represents a higher-order information structure.

In contrast, if we are prepared to loose some information for the sake of better compression, provided the decompressed image data represents a picture that conveys the same message, with acceptable quality, analysis of areas in the picture can help us reconstruct an approximation of the original image using less information.

In *loss-less* (exact) image data compression, as presented in the ISO/IEC 15444 standard [215], which describes both *loss-less* and *lossy* image compression methods, bytes are the symbols in the information stream. Since in typical images many pixels have identical colours, it is possible to derive encoding schemes that yield acceptable compression ratios over noise-less channels. If the communication channels were noisy, then additional error-correcting information might be required for results of acceptable quality.

In this loss-less compression context, it might be largely irrelevant that we wish to compress an image, as the data source is usually treated just the same as if it would have come form any other digital source such as text or audio samples for instance, as discussed in [135] (pp. 193).

As can be expected, this method of compression typically meets with moderate results, since it generally does not take specific advantage of the entropy present in the higher-order information structure[3] represented by the image.

In *lossy* (inexact) image data compression, as also introduced in [215], roughly speaking, image areas are used as the symbols in the information stream. By attempting to find similarities between the various image areas, in terms of colour distribution for instance, it is possible to derive encoding schemes that typically give better compression ratios over the loss-less method. The larger the areas the image is divided into, the poorer the quality of the reconstructed image, and the better the compression ratio. If higher quality results are sought, then the source image must be divided into smaller areas at the expense of a somewhat degraded compression ratio.

File formats using lossy image data compression, such as JPEG [216], are perfectly suited to typical full-colour[4] photographic scenes, with smooth colour gradients and transitions and relatively few sharp edges.

When compressing images using the JPEG [216] scheme, a quality factor can be used to suggest the desired image quality. Typically, the higher the quality factor used, the lower the compression ratio is achieved and vice versa. Since image data is decomposed and analysed in rectangles, sharp edges are likely to be blurred to varying degrees when compressed in JPEG [216] images, depending on the quality setting.

Because JPEG compression requires the use of colour value approximation, to achieve highest possible quality, the resulting images need to have as wide a choice of colours as possible. If the choice of colours were restricted to a small palette, the JPEG [216] approximations would no longer yield good quality results, resulting in striking artefacts introduced into the picture.

Nowhere is this shortfall more striking than in images of monochrome text using small size fonts. As can be seen in Figure 4.5, at the highest possible compression ratio — which unfortunately also represents the lowest image quality factor for this type of compression — the JPEG [216] image file (1019 bytes) is roughly five times larger than the equivalent loss-less image file using GIF (195 bytes) or PNG (175 bytes). In order to alleviate the image corruption by unwanted artefacts, the quality factor would have to be increased, which would lead to even larger image files.

---

[3] It is interesting to note that this pattern of encoding higher-order information in order to achieve better compression results is used by the AudioGraph file format and many other applications, such as MPEG-4 [249]. For instance, when persisting AudioGraph graphical annotations, rather than storing the corresponding image effect, which would require storing the information for all the pixels affected by the annotation, we store the attributes — colour, pen size, the sequence of point coordinates, etc. — that enable us to reconstruct the image algorithmically. For small annotation image sizes, it might be possible for the corresponding raw image data to be smaller than the collection of annotation parameters. However, for typical annotation image sizes, the parametric approach yields consistently better data compression results at the expense of computational demands.

[4] Using 24 bit colour values, with 8 bits for each of the red, green and blue components, also called RGB.

In this situation, the quality of the JPEG [216] image is exceedingly poor and cannot be considered acceptable by comparison with the perfect image reproduction of the GIF [225] and PNG [220] images, both of which introduce no artefacts whatsoever.

Therefore, to achieve excellent quality and high compression ratios for images with a small number of colours — using 1, 2, 4 or 8-bit palettes for 2, 4, 16 and 256 colours respectively — the best candidates seem to be the loss-less compression image compression formats like GIF [225] and PNG [220].

Figure 4.5. Lossy compression may introduce artefacts in textual monochrome images

Since GIF [225] is a proprietary file format and its use in an authoring system would have required the payment of royalties to CompuServe and Unisys [211], its appeal was limited. Being free, and providing several technical advantages over GIF [225] as well, PNG [220] emerged as the most attractive candidate for an image format to be used widely in the AudioGraph system. In the following section, we will take a closer look at the PNG image file format.

4.2.1.1 PNG Image File Format

The Portable Network Graphics — PNG [220] — format was designed to replace the older and simpler GIF [225] format and, to some extent, the much more complex TIFF format [226]. Two major uses of PNG [220] are explored in the following paragraphs: web delivery and image editing.

The PNG [220] image file format, with its choice of loss-less compression scheme, uses the ***zlib*** [224] library, which provides the **`deflate`** algorithm that is directly derived form Shannon's ideas [140]. The average compression ratios obtained by this algorithm range from 2:1 to 5:1,

and it virtually guarantees not to increase the size of the message even in the worst possible cases, unlike the compressor used by the GIF [225] format, which provides no such guarantee.

For web delivery, PNG [220] really has three main advantages over GIF [225]: alpha channels (variable transparency), gamma correction (cross-platform control of image brightness), and two-dimensional interlacing — a method of progressive display. PNG [220] also compresses better than GIF in almost every case, but the difference is generally only around 5% to 25%, not a large enough factor to encourage everyone to switch on that basis alone. One GIF feature that PNG does not try to reproduce is multiple-image support, especially animations; PNG was and is intended to be a single-image format only.

For multi-image support, a very PNG-like extension format called MNG [223] — pronounced *ming* — was finalised in mid-1999 and is beginning to be supported by various applications, but MNG's and PNG's will have different file extensions and different purposes.

For image editing, either professional or otherwise, PNG [220] provides a useful format for the storage of intermediate stages of editing. Since PNG's compression is fully loss-less—and since it supports up to 48-bit true-colour or 16-bit greyscale — saving, restoring and re-saving an image will not degrade its quality, unlike standard JPEG [216] — even at its highest quality settings. Moreover, unlike TIFF [226], the PNG [220] specification leaves no room for implementers to select what features they will support; the result is that a PNG [220] image saved in one application is readable in any other PNG-supporting application.

Note that for transmission of finished true-colour images — especially photographic ones — JPEG [216] can be the better choice, particularly in applications where alpha channels, gamma correction, and two-dimensional interlacing are not essential. Although JPEG's lossy compression can introduce visible artefacts, as shown in Figure 4.5, these can usually be minimised if increases in file size can be tolerated.

For full-colour, photographic images the savings in file size even at high quality levels are typically much better than is generally possible with a loss-less format like PNG [220].

4.2.1.1.1 Compression

PNG's compression is among the best that can be had without losing image information and without paying patent fees, but not all implementations take full advantage of the available power. Even those that do can be thwarted by unwise choices on the part of the user.

PNG supports three main image types: true-colour, greyscale and palette-based — a variable number of palette entries up to 256 are supported, with both colour and greyscale content. Pixel depths of 1, 2, 4, 8, and 16 bits can be specified [221]. JPEG [216] only supports the first two

image types. GIF [225] supports only the third image type — although it can fake greyscale images by using a grey palette.

The unwanted loss of compression performance may come from the ability to mix up image types in PNG. For instance, forcing an application to save an 8-bit palette image as a 24-bit true-colour — also known as RGB, for its red, green and blue components — image will yield a much larger file than strictly necessary. This may be unavoidable if the original has been modified to include more than 256 colours — for example, if a continuous gradient background has been added — but many images intended for web delivery still have 256 or fewer colours.

On the programmer's side, one common mistake is to include too many palette entries in a PNG [220] image. This error is most noticeable when converting tiny GIF [225] images (bullets, buttons, etc.) to PNG format; these images are typically only 1000 bytes or so in size, and storing 256 three-byte palette entries where only 50 are needed would result in over 600 bytes of wasted space.

Another common programmer mistake is to use only one type of compression filter, or to vary them incorrectly. Compression filters, as described in Section 4.2.1.1.2, can make a dramatic difference in the compressibility of the image. However, in general, this is not a feature that users should be forced to experiment with.

Finally, the low-level compression engine itself can be tweaked to compress either better or faster. Usually *best compression* is the preferred setting, but an implementer may choose to use an intermediate level of compression in order to boost the interactive performance for the user. Usually the difference in file size is negligible, but there are rare cases where such a choice can make a big difference.

4.2.1.1.2 Compression Filters

Compression filters are a way of transforming the image data — without losing information — so that it will compress better. Each horizontal line in the image can have one of five filter types associated with it; choosing which of the five to use for each line continues to be considered more of a black art than a science [219]. Nevertheless, at least one reasonably good algorithm is not only known but also described in the PNG specification [220] and implemented in freely available software. Other algorithms are likely to perform even better, but so far, this has not been an active area of research.

Although filters can lead to spectacular results, reaching overall compression ratios of 841:1 in extreme cases, such cases are not at all common.

A more realistic example is the oceanography data at NASA's OceanESIP site [294]. Digital maps displaying various physical measurements can be generated dynamically in either GIF

[225] or PNG format; the PNG [220] versions are invariably one-fifth the size of the GIF's, thanks to PNG's compression filters. For example, a map showing the geotropic current speeds of the north-eastern Pacific Ocean on 15 October 1999 is 48,501 bytes in GIF format but only 9,594 bytes in PNG [220] format.

Using filters, we seek to reduce the number of distinct symbols in the image data, by predicting data based on neighbouring values — to the left, above and left-and-above. If this reduction in the number of symbols is successful then the filtered message entropy is lower and the compression ratio improves. In other words, we will need fewer bits to encode a smaller number of distinct symbols.

4.2.1.1.3 Alpha Channels

Also known as a *mask channel*, an alpha channel is simply a way to associate variable transparency with an image. Whereas GIF [225] supports simple binary transparency — any given pixel can be either fully transparent or fully opaque — PNG [220] allows up to 254 levels of partial transparency in between for typical images (or 65,534 levels for special formats, which are very seldom used apart from fanatical hobbyists).

Although all three PNG [220] image types — true-colour, greyscale and palette — can have alpha information, it is widely used with true-colour images. For such images, instead of storing three bytes for every pixel (red, green and blue), four are required: red, green, blue and alpha, or RGBA. The variable transparency allows us to create *special effects* that will look good on any background, whether light, dark or patterned. For example, a photo-vignette effect can be created for a portrait by making a central oval region fully opaque (i.e., for the face and shoulders), the outer regions fully transparent, and a transition region that varies smoothly between the two extremes. When viewed with a Web browser that supports PNG rendering, the portrait would fade smoothly to white when viewed against a white background, or smoothly to black if set against a black background.

This transparency feature is far more important for the small graphics that are typically used in presentations and on web pages, such as coloured bullet-points and fancy text. Alpha blending makes advanced anti-aliasing possible — creating the illusion of smooth curves on a grid of rectangular pixels by smoothly varying the pixels' colours, as shown in Figure 4.6 — to make rounded and curved images that look good against any background rather than just against a white background.

Given this capability, the same image can be reused in many places without the *ghosting* effect — whitish pixels surrounding the image as a halo —that occurs with GIF's in such circumstances due to the limited capabilities of transparency management in that format.

In order to obtain effective replacements for GIF [225] buttons and icons, they must be comparable in size as well, and that mostly rules out true-colour RGBA images given their size premium.

However, PNG [220] supports alpha information with palette images as well, so an elegant solution exists. A PNG alpha-palette image represents an image whose palette also has alpha information associated with it, not a palette image with a full alpha mask.

In other words, each pixel corresponds to an entry in the palette with red, green, blue and alpha components. Therefore, if we want to have bright red pixels with four different levels of transparency, we must use four separate palette entries to accommodate them — all four entries will have identical RGB components, but the alpha values will be different. If we want all the colours in the palette to have four levels of transparency, the total number of available colours is effectively reduced from 256 to 64. In general, however, only few colours need more than one level of transparency, and determining which ones those should be may become a complex programming challenge. Fortunately, tools such as *pngquant* [222] exist that convert 32-bit RGBA PNG's into 8-bit RGBA-palette images efficiently.

Figure 4.6. Anti-aliased text example.

A particularly useful application of transparency is text anti-aliasing, which is available for the text tool in the version 2.0 of the AudioGraph recorder. Since text annotations are monochrome, we may even provide colour entries for all transparency values if so desired. In practice, 16 or 32 levels of transparency produce very satisfactory results, and excellent size savings. Notice how the anti-aliased text in Figure 4.6 looks rounded and smooth as opposed to the plain text that looks jagged and blocky.

4.2.1.1.4 Gamma Correction

Gamma correction refers to the ability to correct for differences in how computers — and especially computer monitors — interpret colour values. Web authors in particular are probably aware that Macintosh-generated images tend to look too dark on PCs, and PC-generated images tend to look too light on Macs. An image that looks good on an SGI workstation will not look the same on either a Macintosh or a PC, and even a PC-created image would not look right on all PCs.

Gamma information is a partial solution. It is a means of associating a single number with a computer display system, in an attempt to characterise the complex physics at work within a graphics card's digital-to-analogue converter (RAMDAC) and within a cathode-ray tube (CRT) monitor's high-voltage electron gun. Gamma is only an approximation; a better approximation is to use so-called chromaticity values — also supported by PNG [220] — as well as gamma, but even this is an approximation. The absolute best solution currently available is to use a complete colour management system — that, again, PNG supports via the sRGB extension chunk [221]. For most typical uses, however, just supplying the gamma value of the image and correcting for the corresponding gamma value of the monitor system is sufficient.

4.2.1.1.5 Interlacing

Interlacing — or, more generally, progressive display — has been around a long time. GIF [225] has supported it since 1989, TIFF [226] since around the same time — though not in any standardised way — and JPEG [216] since the early 1990s — although it was not widely implemented until 1996. PNG's method is conceptually similar to GIF [225] interlacing and visually similar to progressive JPEG [216] — i.e. two-dimensional.

The first thing to notice is that only the top one-eighth or so of the GIF [225] image is visible by the time the PNG [220] image's first pass is complete. PNG's first pass is only $1/64^{th}$ of the image data; GIF's is $1/8^{th}$. By the time GIF's first pass is done, four PNG passes have been displayed — and unlike the GIF pixels, which are stretched by a factor of 8:1 at this point, the PNG pixels are only stretched by 2:1. Indeed, there is no stretching at all in PNG's odd-numbered passes and its even passes are all stretched by 2:1 vertically. This means that embedded text in an image is typically readable about twice as fast in a PNG image.

Also, note that PNG's seventh pass and GIF's fourth pass are identical — both consist of every other scan-line. They each therefore represent fully one-half of the image data and one-half of the decoding time.

4.2.1.1.6 File Integrity Checks

PNG [220] supports three main types of integrity checking to help avoid problems with file transfers and the like. The first and simplest is the eight-byte magic signature at the beginning of every PNG [220] image. It will detect the most common type of file corruption: that due to the transfer of a binary file in text — or ASCII [165] — mode. On various systems, line-endings in text files may be marked by using a carriage-return character (CR), a line-feed character (LF), or both. For example, Macintoshes use CR's; Unix [207] systems use LF's; and all non-Unix PC systems (DOS, Windows 3.x/95/NT/2000/XP, OS/2) use CR/LF pairs.

PNG's magic signature deftly includes both a CR/LF pair and a single LF. Thus when transferring in text mode to a DOS box, for example, the bare LF will acquire a matching CR; when transferring to a Unix [207] system, the CR/LF pair will turn into a plain LF; and when transferring to a Macintosh, both the CR/LF and the bare LF will probably turn into plain CR's. It's then a simple matter of looking at the first eight or nine bytes in the file to see whether text-corruption occurred — which is exactly the sort of thing the Unix *file* command is designed to do. Although a corrupt signature is not particularly damaging, the real problem consists in the fact that CR and LF characters in the image data — which have nothing in common with line endings or text but instead refer to pixel values or more abstract compressor tokens — will also be converted, thus destroying the image.

The second type of integrity checking is known as a 32-bit cyclic redundancy check or CRC-32. PNG images are divided up into logical data chunks, and each chunk has an associated CRC stored with it. Assuming even a single bit in the chunk changes, the CRC value one would calculate from the damaged data will no longer match the stored CRC value from the original chunk data. This condition can easily be tested for without decoding the image; in fact, it can be tested on the fly, as the image is downloaded, provided the downloading software would be smart enough.

The third type of integrity check applies only to the image-data chunk(s) and is similar to the CRC values. Where an image chunk's CRC value applies to the filtered, compressed data in the chunk, the Adler-32 checksum [219] applies to the complete stream of uncompressed data — regardless of how many image chunks that might span. It is really only used by the lowest-level compression library as a check against bad encoding and decoding software.

4.2.1.1.7 Pronunciation

Such a detail has been a significant topic of discussion among the authors of the format. The explanation for this peculiarity lies in the awkward situation faced by the GIF [225] format. It is sometimes pronounced with a soft G like *giraffe*, and at other times with a hard G like *gift*, and no one really knows what the correct pronunciation is supposed to be or what the interlocutor

might be referring to when she is using a different pronunciation than one is accustomed to. It appears that the soft G variation — as in *giraffe* — should be considered correct, as it is how the author of the format pronounces it.

PNG is always spelled *PNG* (or *Portable Network Graphics*) and always pronounced *ping* in English, not *pinj* or *pee en gee* or any other multi-syllabic offensive atrocity [219]. For non-English speakers however, the three-letter pronunciation seems to be acceptable to the authors. If in doubt, you may always consult the introduction to the PNG [220] specification for the definitive statement on the matter.

4.2.2 The Challenges of Speech Compression

The human voice typically has most of its useful signal between a few hundred Hz and 4 kHz [241]. As a result, to achieve accurate reproduction, voice signals must be sampled at a rate of at least 8 kHz, according to Nyquist's theorem [239].

For economy, using 8-bit samples (which give a relatively poor sound quality), sampled at a frequency of 8kHz, we need 8000 bytes per second for the voice signal. At a pessimistic 2:1 compression, this signal would require 32 kbps of bandwidth, which is more than twice of what is available over a 14.4 kbps modem connection, considered typical at the time we started our work, circa 1997. At a hypothetical 5:1 compression, we might just achieve 12.8kbps, but given the typical characteristics of voice sound signals, such high compression ratios cannot be guaranteed with traditional loss-less compression methods.

In addition, in order to get better sound reproduction we could strive to use more bits per sample, such as 16-bit samples for instance. However, this would double our bandwidth requirements, which is unacceptable under our self-imposed economy regime. In such a scenario, for encoding speech, a typical loss-less compression ratio between 2:1 and even a very optimistic 5:1 would not be quite sufficient to stream the signal over a low bandwidth connection, although loss-less data transmission would be achieved — and therefore 'perfect' data transmission quality could be available.

In the case of a voice stream, communication is usually parsed in terms of recurring sound patterns — *phonemes*, the smallest phonetic units in a language capable of conveying a distinction in meaning — rather than any digital representation of the signal, such as bytes. Therefore, choosing bytes as the symbols in the information stream for digitised voice signals is likely to be a rather poor choice as a starting point for compression, as they are unlikely to match the sound patterns with any regularity.

In this specific context, a different approach is required to compress voice signal streams effectively, and with high enough quality. Fortunately, a lot of research in this area has been

done since the telecommunications industry is in the business of transmitting very good quality voice signals with the least bandwidth expenditure possible.

In an analogy with the lossy image compression schemes, it has proved fruitful to seek to determine the actual *speech symbols* (phonemes) in the voice information stream. Then, even though the decoded signal was only an approximation of the source signal, the compression ratio achieved could be significantly higher. Given enough care in the process, excellent voice quality could be achieved. This is exactly the approach used by the Regular-Pulse Excitation Long-Term Predictor algorithm (RPE-LTP [241]) we have chosen for the AudioGraph suite of applications.

We considered the 14.4 kbps bandwidth target — satisfied by RPE-LTP's 12.8kbps — a worthy goal at the time that this project was started (1997), since we estimated that modems that supported at least such connection speeds if not higher would be widely available at low cost in most markets. Therefore, we could provide a viable web-based lecturing solution without requiring students to invest in a significantly more expensive Internet connection — such as cable modem or ADSL.

4.2.2.1 GSM sound codec — RPE-LTP

The speech compression algorithm used by the GSM suite is called GSM 06.10 RPE-LTP [227]. We have chosen to use this speech compression method as it offers an excellent compression ratio of slightly over 9 to 1, with 13 bit samples, as presented in more detail in a number of our papers such as [6], [9] and [15], and further along in this section.

The implementation for the GSM compression and decompression library we use was developed Jutta Degener at the Technical University of Berlin. At the time, we were designing and developing the AudioGraph suite of applications (1997), the GSM 06.10 [227] compression was thought to be available without having to pay royalty fees.

As history has shown, with Fraunhofer IIS [248] and THOMSON Multimedia's demands for royalty payments [253] for applications recording MPEG-1 Layer 3 (more commonly known as MP3 [252]), the situation seems to have changed for GSM 06-10 [227] as well as now Philips is claiming intellectual property on it (patent #4932061). However, at the time of writing, Philips had not yet approached either the authors of the library we use, nor the organization that now markets the AudioGraph software [3].

In fact, we might find ourselves lucky, as we happen to use a variant of the GSM 06-10 [227] compression, as supported natively by Windows as the .wav chunk format #49, and used by Microsoft in the context of VPIM — Voice Profile for Internet Mail. Microsoft had submitted this format to the IETF for consideration [240], but that submission has expired as of March 20,

2000. The VPIM IETF workgroup was actually looking at using this format. It is yet unclear whether Philips has released or not their intellectual property for use in VPIM.

In any case, in the near future the AudioGraph will offer as an alternative the Ogg Vorbis [242] audio compression format, which aims to provide a royalty-free algorithm that would rival MP3 [252] in quality and compression ratios, and tuneable streaming capabilities from 16 to 128 kbps per channel.

Jutta Degener gave an excellent introduction to the principles of the GSM 06-10 [227] compression algorithm in an article in Dr. Dobb's Journal [241]. In the following sections, we follow Jutta's ideas, with a few clarifying comments.

4.2.2.1.1 Human Speech

Phonetics classifies speech as being primarily made up of *voiced* and *unvoiced* sounds and goes on to define various sub-categories of sound (fricative/affricate/plosives/liquids/nasals and so on). For the purposes of RPE-LTP sound compression, we only need to concern ourselves with the distinction between voiced and unvoiced sounds.

In voiced sounds, the vocal cords contribute to sound production. Predictably, the vocal cords play no part in producing unvoiced sounds. RPE-LTP is mostly concerned with voiced sounds, and is most suitable for languages that most frequently use voiced phonemes. English happens to be one of these languages.

When voiced speech is pronounced, air is pushed out from the lungs, opening a gap between the two vocal folds (also called the *glottis*). Tension in the vocal folds (or cords) increases until — pulled by muscles and the Bernoulli force from the stream of air — they close. After the folds have closed, air from the lungs again forces the glottis open, and the cycle repeats — between 50 to 500 times per second, depending on the physical construction of the larynx and how strong the pull on the vocal cords is.

For unvoiced consonants, we blow air past some obstacle in our mouths, or let the air out with a sudden burst. Where we create the obstacle depends on which atomic speech sound — also called *phoneme* — we wish to make. During transitions, and for some *mixed* phonemes, we use the same air-stream twice — first to make a low frequency hum with our vocal cords, then to make a high frequency noisy hiss in our mouths. Some languages, such as Eskimo and many Indian dialects, have complicated clicks, bursts, and sounds for which air is pulled in rather than blown out. Such cases are not described well by this model.

We never really hear someone's vocal cords vibrate. Before vibrations from a person's glottis reach our ears, those vibrations pass through the throat, over the tongue, against the roof of the mouth, and out through the teeth and lips.

The space that a sound wave passes through changes it; as parts of one wave are reflected, they mix with the next oncoming wave, changing the sound's frequency spectrum. Every vowel has three to five typical — *formant* — frequencies that distinguish it from others. By changing the interior shape of our mouths, we create reflections that amplify the formant frequencies of the phoneme we are speaking.

4.2.2.1.2 Digital Signals and Filters

To represent a sound wave digitally, we have to sample it at a certain frequency and the samples must be digitised. According to Nyquist's theorem [239], the sampling frequency must be at least twice as high as the highest frequency in the wave. Speech signals, whose interesting frequencies go up to 4 kHz, are often sampled at 8 kHz and digitised on either a linear or logarithmic scale — μ-Law compression [233] uses a logarithmic scale for instance.

The input to GSM 06.10 [227] consists of frames of 160 signed, 13-bit linear PCM [234] values sampled at 8 kHz. One frame covers 20 ms, about one glottal period for someone with a very low voice, or ten for a very high voice. This is a very short time, during which a speech wave does not change too much. The processing time plus the frame size of an algorithm determines the *transcoding delay* of our communication.

The encoder compresses an input frame of 160 samples to one frame of 260 bits. One second of speech turns into 1625 bytes; a megabyte of compressed data holds a little more than ten minutes of speech.

The .wav #49 variant we use in the AudioGraph system actually uses double-size frames, compressing 320 13-bit samples to 520 bits — precisely 65 bytes. It is slightly more efficient than the plain GSM 06-10 [227] alternative, since 260 bits actually represent 32 bytes and 4 bits, which the library packs in 33 bytes for the compressed frames, effectively wasting a nibble in each frame.

The concept of *filter* is central to signal processing. A filter's output can depend on more than just a single input value — it can also keep state. When a sequence of values is passed through a filter, the filter is *excited* by the sequence. The GSM 06.10 [227] compressor models the human-speech system with two filters and an initial excitation. The linear-predictive short-term filter, which is the first stage of compression and the last during decompression, assumes the role of the vocal and nasal tract. It is excited by the output of a *long-term predictive* (LTP) filter that turns its input — the *residual pulse excitation* (RPE) — into a mixture of glottal wave and voiceless noise.

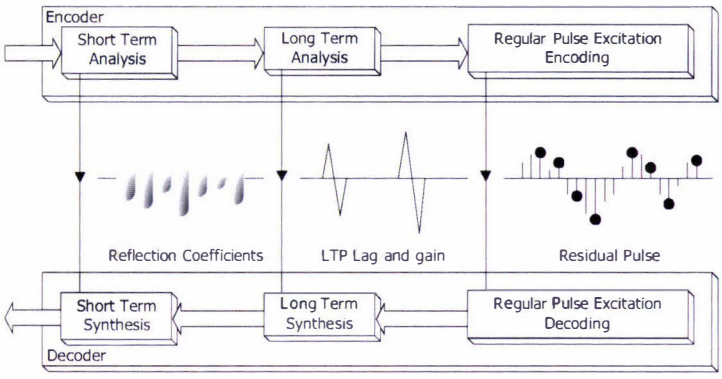


Figure 4.7. Overview of the RPE-LTP compression scheme

As Figure 4.7 illustrates, the *encoder* divides the speech signal into short-term predictable components, long-term predictable components, and the remaining residual pulse. It then breaks the signal into samples and encodes the pulse and parameters for the two predictors. The *decoder* — the synthesis section — reconstructs the speech by passing the residual pulse through the long-term prediction filter, and forwards the output of that stage through the short-term predictor.

4.2.2.1.3 Linear Prediction

To model the effects of the vocal and nasal tracts, such a filter design is required that, when excited with an unknown mixture of glottal wave and noise, it produces the speech we are trying to compress. By restricting ourselves to filters that predict their output as a weighted sum — or *linear combination* — of their previous outputs, it becomes possible to determine the sum's optimal weights from the output alone. The filter's output, the speech, is the algorithm's input.

For every frame of speech samples, we can compute an array of weights so that the signals are as close as possible to a linear combination of the previous sample frames for all sample values. The number of previous sample frames taken into account is usually between 8 and 14. GSM 06.10 [227] uses eight weights.

The results of GSM's *linear-predictive coding* (LPC) are not the direct linear combination coefficients of the filter equation, but the closely related reflection coefficients.

When a wave arrives at the boundary between two tubes of different diameters, it does not simply pass through — parts of it are reflected, and interfere with waves approaching from the back. A reflection coefficient calculated from the cross-sectional areas of both tubes expresses the rate of reflection. The similarity between acoustic tubes and the human vocal tract is not accidental; a tube that sounds like a vowel has cross-sections similar to those of our vocal tracts when we pronounce that vowel. However, the walls of our mouths are soft and absorbent, not loss-less, and in consequence, some wave energy is converted into heat. The walls are also not

entirely rigid either — they vibrate and resonate with lower frequencies. If the connection to the nasal tract is opened as well — which is usually closed in most cases for most western languages — the model will not resemble the physical system very much. Therefore, *nasal* sounds are likely to be reproduced less accurately through this encoding scheme.

4.2.2.1.4 Short-Term and Long-Term Analysis

The short-term analysis section of the algorithm calculates the short-term residual signal that will excite the short-term synthesis stage in the decoder. We pass the speech backwards through the vocal tract by almost running backwards the loop that simulates the reflections inside the sequence of loss-less tubes. The remainder of the algorithm works on 40-sample blocks of the short-term residual signal, producing 56 bits of the encoded GSM frame per iteration.

The LTP analysis selects a sequence of 40 reconstructed short-term residual values that resemble the current values. LTP scales the values and subtracts them from the signal. As in the LPC section, the current output is predicted from the past output.

The prediction has two parameters: the LTP lag, which describes the source of the copy in time, and the LTP gain, the scaling factor. To compute the LTP lag, the algorithm looks for the segment of the past that most resembles the present, regardless of scaling. Resemblance is computed by correlating the two sequences whose resemblance we want to know.

The correlation of two sequences, x[] and y[], is the sum of the products x[n]*y[n-lag] for all n, and it is a function of the lag, of the time between every two samples that are multiplied. The lag between 40 and 120 with the maximum correlation becomes the LTP lag. In the ideal voiced case, that LTP lag will be the distance between two glottal waves, which is the inverse of the speech's pitch.

The second parameter, the LTP gain, is the maximum correlation divided by the energy of the reconstructed short-term residual signal. The energy of a discrete signal is the sum of its squared values — its correlation with itself for a lag of zero. We then scale the old wave by this factor to get a signal that is not only similar in shape, but also in loudness.

4.2.2.1.5 Residual Signal

To remove the long-term predictable signal from its input, the algorithm subtracts the scaled 40 samples. We hope that the residual signal is either weak or random and consequently cheaper to encode and transmit. If the frame recorded a voiced phoneme, the long-term filter will have predicted most of the glottal wave, and the residual signal is weak. If the phoneme was voiceless, the residual signal is noisy and does not need to be transmitted precisely. Because it cannot squeeze 40 samples into only 47 remaining GSM 06.10-encoded bits, the algorithm down-samples by a factor of three, discarding two out of three sample values. We have four

evenly spaced 13-value sub sequences to choose from, starting with samples 1, 2, 3 and 4. The first and the last have everything but two values in common.

The algorithm picks the sequence with the most energy—that is, with the highest sum of all squared sample values. A 2-bit *grid-selection* index transmits the choice to the decoder. That leaves us with 13 3-bit sample values and a 6-bit scaling factor that turns the PCM [234] encoding into an APCM — Adaptive PCM [235]; the algorithm adapts to the overall amplitude by increasing or decreasing the scaling factor.

Finally, the encoder prepares the next LTP analysis by updating its own remembered *past output*, the reconstructed short-term residual. To make sure that the encoder and decoder work with the same residual, the encoder simulates the decoder's steps until just before the short-term stage. It deliberately uses the decoder's grainy approximation of the past, rather than its own more accurate version.

4.2.2.1.6 The Decoder

Decoding starts when the algorithm multiplies the 13 3-bit samples by the scaling factor and expands them back into 40 samples, zero padding the gaps. The resulting residual pulse is fed to the long-term synthesis filter: The algorithm cuts out a 40-sample segment from the old estimated short-term residual signal, scales it by the LTP gain, and adds it to the incoming pulse. The resulting new estimated short-term residual becomes part of the source for the next three predictions.

Finally, the estimated short-term residual signal passes through the short-term synthesis filter whose reflection coefficients the LPC module calculated. The noise or glottal wave from the excited long-term synthesis filter passes through the tubes of the simulated vocal tract—and emerges as speech.

4.2.2.1.7 Conclusion

As we have seen, RPE-LTP [241] finds an ingenious way to compress voice signals by identifying higher-level patterns in the sound stream. Although this is a lossy compression scheme, the subjective quality provided for voice signals is excellent, providing effortless speaker identification.

4.3 Design Patterns

In a purposefully simplified view, one may state that once a good conceptual design is established, object-oriented software construction might be considered to become a matter of routine. One simply has to follow the directions provided by the various design elements: interfaces, classes, methods and attributes.

However, in real life, since we generally have to deal with substantial complexity, object-oriented software design is quite difficult. Suitable objects must be identified in the problem domain; they must be organised into classes with appropriate granularity for good reuse and maintainability properties; attributes must be defined in the correct places in the class hierarchy; good inheritance hierarchies must be defined, and effective interfaces and relationships must be established between classes.

Therefore, in order to cope with the complexities of object-oriented software design and learn how to deal effectively with these challenges, we need to develop and use a higher level of abstraction beyond mere design elements like classes, methods, attributes and inheritance hierarchies. This higher level of abstraction is the realm of *design patterns*.

4.3.1 Introduction to Design Patterns

Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides make an excellent introduction to the subject of *design patterns* in their famous book [127], now widely known among computer professionals by the affectionate nickname "the gang-of-four book".

In seeking to define what a *design pattern* is, they quote the renowned architect Christopher Alexander: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" [261].

Although Alexander refers to patterns in architecture and urban planning, the statement applies equally well in the software field, to object-oriented design patterns. As the "gang-of-four" suggest [127] (pp. 3), software design patterns can be described through four essential elements:

- **Pattern name**. It is a means of identifying the design problem, its solutions and associated trade-offs in a concise fashion.
- **Problem**. Clarifies when the pattern should be applied, and describes the usual contexts for which it is appropriate.
- **Solution**. Under this heading are presented the responsibilities, relationships and collaborations of the elements that make up the solution design.
- **Consequences**. This section describes the benefits and drawbacks that derive from the application of the pattern. The considerations outlined in this section are useful in evaluating when and how patterns are best applied.

In the following sections, we will highlight various design patterns or their variations that we have made use of, explain the benefits we have gained by using them, and present various trade-

offs that we have faced. In each section, we will briefly describe the design pattern we are focusing on by covering its four essential elements.

4.3.2 Terms

Before we proceed, it is important to define a number of terms that we shall use throughout the following sections.

The term *class hierarchy* refers to the design hierarchy of classes, as dictated by their inheritance relationships (is-a). Classes having a common ancestor are said to belong to the same class hierarchy.

The term *structural hierarchy* refers to the run-time hierarchy of instances, as realised through composition relationships (has-a). Object instances in structural hierarchies may be realizations of classes from different class hierarchies.

4.3.3 Composite

4.3.3.1 Problem

In many instances, we deal with objects belonging to the same class hierarchy that need to be assembled in arbitrarily complex structural hierarchies. As a result, it becomes necessary to use some objects as abstractions for collections of objects.

4.3.3.2 Solution

Make some objects capable of being containers that host other object instances — which may be containers themselves, or may be leaf objects. In this fashion, the container objects may then forward the base-class operations to all the contained objects, giving birth to the *composite* design pattern [127] (pp. 163).

Complex structural hierarchies can be defined, retaining the uniform access interface, and therefore this approach is very useful in managing complexity.

4.3.3.3 Consequences

Clients of the *composite* design pattern can treat structural hierarchies of composites and leaves uniformly. In the context of this design pattern, it is simple to add new classes to the class hierarchy and retain the uniformity of access.

As a drawback, since the composite containers accept all leaves and composites, it may become difficult to restrict the composition of a container at compile-time.

4.3.4 Iterator

4.3.4.1 Problem

Containers are at the heart of computation. In countless instances, it is necessary to perform computations on some or all of the elements in a container. On traditional sequential computers, iterating through the items in the container and performing the desired processing in the iteration loop achieves the desired results.

If multiple clients happen to iterate through the same container at the same time, it may also become important to notify all clients should the contents of the container change, which PowerPlant [148] implements in its **`LArray`** and **`LArrayIterator`** classes and their associated template classes. In this respect, in order to satisfy strict performance requirements, the Standard Template Library — STL [143] — takes a more drastic approach than PowerPlant, and specifies that certain operations that affect the contents of containers may invalidate all active iterators [143] (pp. 130).

4.3.4.2 Solution

When we wish to retrieve information from a container, we could either use special methods provided by the container class to access the contained items directly, or we could use an indirect approach, using a special kind of 'pointer' object to the elements hosted by the container, called an *iterator* [127] (pp. 257).

Iterators are classes that provide such 'pointer' functionality for the various container classes. Typically, each container class implements its own specific types of iterators. In standard C++, container classes will generally implement both an **`iterator`** and a **`const_iterator`** type, as do all containers in the STL [143].

4.3.4.3 Consequences

Using iterators presents more flexibility in data access compared to direct access to the data. For instance, multiple non-destructive traversals can occur in parallel over the same container, since the iteration state is kept with the iterator, not the container. In C++, iterators can even be optimised to present no overhead compared to calling direct access methods on containers, as suggested by Coplien [142] (pp. 163).

Some libraries, like the STL [143], now part of the C++ standard [134], do not require iterators to behave gracefully in the presence of change in the container, choosing simply to invalidate all other iterators when a change occurs.

However, the PowerPlant [148] LArrayIterator class arranges to notify all iterators pointing to the same container of any change in the container, and makes them adjust their state accordingly — implemented using the *observer* design pattern [127] (pp. 293).

4.3.5 Memento

4.3.5.1 Problem

Modern software systems are becoming increasingly complex, since they have to model increasingly richer systems. It is therefore very useful to be able to capture the state of a complex software system, stream it into a persistent form, and afterwards restore the structural hierarchy from it.

On a more abstract level, it is also desirable to be able to capture and persist the state of an object so that it might be restored if required. This capability may prove useful in implementing *undo* functionality, transaction rollback, and so on.

4.3.5.2 Solution

Persistent objects will expose an interface that allows them stream themselves into a persistent form or create themselves from a persistent description, all without violating encapsulation. This is the essence for the *memento* design pattern [127] (pp. 283).

4.3.5.3 Consequences

This design pattern is invaluable in building complex visual hierarchies very quickly and efficiently. Most user interfaces require myriad controls, views and other visual elements, and therefore it has become standard practice in most modern programming environments to be able to define user interface components using a modelling tool, and then having the software framework replicate the user interface from the description recorded by the modelling tool.

The *memento* design pattern [127] (pp. 283) ensures that encapsulation boundaries are preserved, as persistent objects do not have to expose their internal details in the persistence process.

Making extensive dynamic use of *mementos* may become inefficient, particularly if large amounts of information need to be preserved in order to capture state. Clients of persistent objects must also assume responsibility for the *mementos*, since the persistent objects themselves are not required to keep track of them. They are merely required to be able to store snapshots of their state and re-create them on demand.

4.3.6 Template Method

4.3.6.1 Problem

Code reuse is a worthwhile goal in modern software development, given that it promises to increase the robustness of the system and decrease development time. In certain cases, in order to maximise code reuse, we may wish to reuse the basic structure of an algorithm, delegating specific implementations of the various steps to subclasses.

4.3.6.2 Solution

We may achieve the above goal by capturing the algorithm we wish to reuse in a single method — called *template method* [127] (pp. 325) — and providing calls to primitive operations that subclasses could be expected to provide in order to achieve the desired functionality. Typically, the class implementing the template method is abstract, exposing the primitive operations as pure virtual methods.

4.3.6.3 Consequences

The *template method* design pattern [127] (pp. 325) is one of the fundamental techniques for code reuse, having a very important part to play in class libraries and application development frameworks, factoring out common behaviour patterns while providing flexibility.

The use of template methods leads to an inverted control structure, since the parent class orchestrates the operations of subclasses, and not the other way around, as is otherwise frequently the case in object-oriented designs.

Hook methods are an interesting variant of method that may be invoked in the *template method* design pattern. As opposed to *abstract* methods, which **must** be implemented, *hooks* are methods that **may** be implemented by derived classes to provide extra functionality at clearly defined points in the execution sequence.

By decoupling *hooks* into object instances, a variant of the *decorator* design pattern [127] (pp. 175) is realised, such as it is provided in the PowerPlant framework [148], in the guise of the `LAttachable` and `LAttachment` classes, as discussed in Section 4.3.10.

4.3.7 Strategy

4.3.7.1 Problem

In some contexts, we may wish to be able to use a family of algorithms interchangeably. That is, we may require algorithms to vary independently from the clients that use them.

4.3.7.2 Solution

In order to accomplish this, we encapsulate each algorithm and make them all present the same interface. As a result, we can specify which algorithm is to be used by simply instantiating the desired class.

The *strategy* design pattern [127] (pp. 315) is similar to the *template method* design pattern, except that the whole algorithm is encapsulated in the strategy class. As a result, algorithms may vary substantially, not being required to supply variants for specific types of operations.

4.3.7.3 Consequences

As strategy classes can be organised in hierarchies, inheritance may be used to factor out common functionality among the related algorithms. Using the *strategy* design pattern [127] (pp. 315) can also offer an appealing alternative to sub-classing in certain instances by making it much easier to understand the context in which multiple algorithms may be executed. The alternative, that of encapsulating each algorithm in a sub-class of the context itself is much more brittle and significantly harder to extend.

Strategies can be used to eliminate conditional statements. It is this characteristic of the *strategy* design pattern [127] (pp. 315) that has drawn us to it on a number of occasions, primarily for its performance benefits. If support for different behaviours required by an application is provided as part of the same class, then typically some form of conditional statement will be required to select the appropriate functionality.

Using the *strategy* design pattern [127] (pp. 315), no further conditional statements are required, as the strategy is already known, and only an indirection invoking the appropriate algorithm is required.

Strategies may also provide different implementations for the same overall behaviour. Clients may therefore choose between various alternatives with different time and space constraints, or with different side effects such as providing profiling or monitoring information.

The fact that the client component must understand the implications of using each strategy before it can select the appropriate one may be interpreted as a potential drawback, since it may expose clients to implementation details. Therefore, it is imperative that this design pattern be used carefully, without unnecessarily exposing implementation issues.

If certain strategy algorithms do not make full use of the context information passed to them, the communication overhead between the context in which the strategy is executed and the strategy algorithm may become a penalty, as dictated by usage patterns. However, this drawback can be

mitigated in certain circumstances by allowing strategies to extract the information they require on demand.

Another potential drawback of using strategies is that the number of objects in the application may rise if the strategies are associated with contexts on a per-instance basis. This effect may be mitigated by providing the strategies as stateless objects that can be shared among the various contexts of execution.

4.3.8 Builder

4.3.8.1 Problem

We may sometimes find ourselves faced with the problem of having to load/save documents containing collections of components of known types from/to a variety of file formats. Ideally, we would like to be able to reuse as much as possible of our infrastructure when doing so.

4.3.8.2 Solution

The *builder* [127] (pp. 97) design pattern makes use of a *director* that orchestrates the building process and a *concrete builder* that knows how to construct or persist the various components manipulated by the system.

These two elements communicate through an abstract interface that aggregates the methods for loading and saving each kind of component supported by the document. In C++, this is achieved by defining an abstract base class with methods such as:

```
virtual void        Load(CEpisode &inEpisode) = 0;
virtual void        Save(CEpisode &inEpisode) = 0;
virtual void        Load(CLineComponent &inComponent) = 0;
virtual void        Save(CLineComponent &inComponent) = 0;
```

And so on. The abstract base class in the *builder* design pattern [127] (pp. 97) may also provide a means to orchestrate the invocation sequence for these methods — resembling the *template method* design pattern [127] (pp. 325).

However, *builder* is a creational design pattern, concerned primarily with creation and persistence of objects, whereas *template method* is a behavioural, more generic design pattern. Given this arrangement, concrete instance classes need only provide implementations for the abstract methods to resurrect/externalise the structural hierarchy.

Therefore, we can provide a builder class for each supported file format, and we can dynamically specify the adequate builder for the particular type of file we need to deal with at the time.

4.3.8.3 Consequences

It is possible, using this design pattern, to vary the internal representations of the various components independently, as required. Since the *concrete builder* presents an abstract interface to the *director*, the builder may hide the internal structure of the component being constructed.

All that would be required for changing the component's internal representation, should this be required, would be the definition of a new kind of builder that supports it.

This approach also separates the complexities of building a non-trivial structural hierarchy from its representation. Client code requires no knowledge of the specific classes required to build the desired structure, since concrete builder classes contain all the information required to construct a particular kind of structural hierarchy.

The *builder* design pattern offers better control over the construction process, since it requires the orchestration of a number of components rather than providing finished products all at once, like other creational design patterns such as *factory method* [127] (pp. 107).

4.3.9 Singleton

4.3.9.1 Problem

In certain instances, it is desirable to ensure a single instance of a particular class exists within a particular application.

4.3.9.2 Solution

We create a class with a private constructor, which exposes a public static member function that returns the unique instance of the class, called a *singleton* [127] (pp. 127).

4.3.9.3 Consequences

The problems caused by the unknown order of initialisation of static variables from different compilation units can be alleviated by carefully orchestrating the use of singletons in instances where interdependencies between them may exist.

The *singleton* design pattern [127] (pp. 127) promotes controlled access to the instance, and is a significantly better alternative to a global variable that would pollute the global namespace.

If the *singleton* interface is defined carefully, sub-classing it may prove very useful, providing applications with the ability to configure themselves at run-time with a specific set of desired capabilities exposed through adequate singleton subclass instances.

Should the restriction on uniqueness need to be relaxed, using the *singleton* framework makes it very easy to control precisely the cardinality of the population of instances.

Using a *singleton* instance and invoking methods on it rather than using static member functions provides more flexibility, since the latter cannot be *virtual* and therefore the benefits of sub-classing and simple control of the instance population cannot be easily realised. However, if these features are not required by a particular implementation, static member functions may prove slightly more efficient.

4.3.10 Decorator

4.3.10.1 Problem

With a defined structural hierarchy in place, it is sometimes necessary to add special processing abilities at runtime, at different points in the structural hierarchy. Similarly, we may wish to dynamically assign certain particular responsibilities to specific object instances rather than to whole classes.

4.3.10.2 Solution

The *decorator* design pattern, as described in [127] (pp. 175) could prove a good candidate for solving this type of problem. The PowerPlant [148] framework provides a variant called *attachment* that uses a similar mechanism.

Whereas the *decorator* design pattern suggests that the class extending a particular piece of functionality should act as a lightweight wrapper, using a variation of the envelope-letter idiom described by Coplien [142] (pp. 133), the *attachment* mechanism takes a composition approach.

In PowerPlant, various classes derive from **`LAttachable`**. Instances of such classes may accept one or more class instances derived from **`LAttachment`**. When processing the various messages flowing through the structural hierarchy of applications, *attachments* are given the opportunity to intervene, achieving effectively capabilities virtually identical to those of a *decorator*. *Attachments* also have the ability to specify that the message handler that invoked them should not execute, indicating that the message has been fully handled by the *attachment*.

4.3.10.3 Consequences

Where *decorators* intercept calls to specific member functions and 'decorate' them as appropriate with their specific functionality, the *attachments* rely on the framework components for being invoked.

In the context of a framework such as PowerPlant [148], where various library classes or their descendants are very likely to be represented in the actual structural hierarchy, the *attachment* approach provides a more disciplined approach, with less impact on the structural hierarchy.

In the *decorator* design pattern [127] (pp. 175), although the *decorator* instance acts as a transparent enclosure for its *decorated* component, they are not identical from an object identity point of view. In the *attachment* approach, the *decorated* object stands for itself in the larger structural hierarchy, without any wrappers, no matter how transparent. It is interesting to note that the use of the *attachment* mechanism does not preclude the use of the *decorator* design pattern however, as they are orthogonal.

The *decorator* design pattern [127] (pp. 175) and its *attachment* variant provide better flexibility in dynamically assigning responsibilities between object instances than would be easily possible by using multiple-inheritance. While using the *decorator* approach new responsibilities can be configured dynamically at runtime, multiple-inheritance would require the definition of a new class to achieve a similar effect. It is also possible to provide instances of the same *attachment* type multiple times if so desired, whereas inheriting directly from the same class multiple times is not viable.

By using the *decorator* design pattern [127] (pp. 175), it is possible to design simple, elegant classes without sacrificing flexibility. Rather than attempting to design a heavy class that would attempt to provide for any foreseeable extension or capability, specialised functionality may be added as required through *decorators* or *attachments*.

However, the overuse of *attachments* or *decorators* may lead to systems that are harder to understand and debug, as special pieces of functionality become 'hidden' away form the main body of code, which may well not even get to execute if so dictated by the *decorator*.

4.3.11 Command

4.3.11.1 Problem

Current graphical user interfaces are generally driven through a menu interface. Users select menu items, which in turn invoke various commands to perform the desired actions. The same menu item may apply to different target objects at different points in time.

For instance, the *Copy* menu item in the *Edit* menu will always refer to the current selection of elements, such as episodes or annotations. Since the selection changes in response to user actions, the effect of the *Copy* command must be adjusted accordingly.

If the classes that implement the menu functionality were to be directly responsible for performing the required actions as well, then specific knowledge associated with the target objects would have to be incorporated into them, in effect violating encapsulation. In such a scenario, each time a new type of annotation would be added to the system, the menu classes would need to be updated to recognise it and know how to deal with it, which is undesirable.

Also, in most modern applications, multiple undo support is required, since the forgiveness imparted by its availability is very useful from a user-interface design perspective. Therefore, we must find a way to capture user actions in such a manner as to allow us to provide the ability to undo and redo their effects at will.

4.3.11.2 Solution

We encapsulate user actions into objects in their own right, so that the action invocation and its execution can be decoupled. This approach also makes it possible to undo/redo multiple actions in a controlled sequence.

4.3.11.3 Consequences

The *command* design pattern [127] (pp. 233) facilitates decoupling the instance that invokes an operation from the one that will perform it. As a result, since commands are first-class objects in their own right they may be extended as required, and manipulated as any other instances.

This capability is particularly useful when defining an undo stack, as commands may be stored and accessed later on as required.

Another similar use could be to aggregate multiple commands into composite commands, potentially through an instantiation of the *composite* design pattern [127] (pp. 163).

Since *commands* do not require any existing component classes to be modified, it is generally very easy to define new ones.

4.3.12 Observer

4.3.12.1 Problem

Sometimes, the state of a particular object instance may be of interest to multiple associated objects. All dependents will wish to be notified when this object changes state.

4.3.12.2 Solution

The solution is to provide a mechanism through which a *listener* object can register itself with *broadcaster* objects, whose state is of importance to the *listener*. Then, as state changes in the *broadcaster* happen, a notification will be sent to all registered *listeners*. In effect, the *listeners* 'observe' the states of the *broadcasters*, hence the name of this design pattern, *observer* [127] (pp. 293).

4.3.12.3 Consequences

The main benefit of the *observer* design pattern lies in the fact that the numbers of *listeners* and *broadcasters* and their interconnections may be varied independently. As a result, multiple *listeners* can listen to the same *broadcaster* and vice versa, and their relationships may be established and updated dynamically.

The abstract coupling between *broadcasters* and *listeners* means that broadcasters only need to know that they may have a number of *listener* clients, each conforming to the abstract *observer* interface, which makes for minimal interdependencies.

This neutral, simple interface allows *broadcasters* and *listeners* belonging to different layers of abstraction within the system to communicate directly, without compromising the layering abstraction in any way.

The very nature of this design pattern is well suited for broadcast communication. Therefore, unlike typical messaging schemes that require a known destination, the *observer* design pattern facilitates a *publish-subscribe* messaging paradigm, with messages from *broadcasters* reaching all interested *listeners*. This lack of concern with message destinations makes it possible to manage dynamically the list of *listeners*.

A potential drawback of this design pattern lies in the fact that *listeners* are generally oblivious to the presence of any other siblings in a *broadcaster's* collection of *listeners*. In certain cases, this situation may lead to unexpected effects, particularly if the *listeners* happen to have overlapping responsibilities. The situation may be even further complicated by the fact that generally the notification protocol is too simple to clarify what component of the *broadcaster's* state has changed, possibly requiring *listeners* to investigate such details.

4.4 Development Frameworks

As technological capabilities of computers have dramatically improved over the past twenty-five years, the user expectations for user-friendly graphical user interfaces have soared as well.

Since most applications must rely on similar controls and user interface components to build their graphical user interfaces, in order to minimise user confusion, application vendors might be required to develop independently very similar code if they had no help from standardised user interface controls libraries. The need for low-level development is almost certain to increase the overall development time and potentially introduce substantially more defects in the end product. If application developers must spend lots of time and effort in defining and assembling the user interface components, it will be difficult for them to focus on the core functionality that their application is supposed to provide.

Since the shortest possible delivery time and the highest robustness are two of the major goals of virtually all software development projects, and both conflict with the complexity of un-aided graphical user interface development, significant efforts have been made to develop application development frameworks that would assist in meeting those goals [149].

Gamma et al. [127] (pp. 26) discuss the concepts of *toolkits* and *frameworks* in the context of reuse mechanisms. They regard *toolkits* [127] (pp. 362) as collections of classes that provide useful functionality, but do not necessarily fully define the design of an application, while *frameworks* [127] (pp. 360) are sets of cooperating classes that facilitate reusable design for a particular class of software. A *framework* provides architectural guidance by crystallising the responsibilities and collaborations of relevant abstract and concrete classes that cover a certain design space. Users may customise the *framework* for a particular purpose by deriving from and/or composing *framework* class instances, and extending the skeleton thus provided to cover the necessary application space.

Generic graphical user interface application development *frameworks*, such as PowerPlant [148] provided by Metrowerks with its CodeWarrior [147] integrated development environment tools for the Macintosh platform and the Microsoft Foundation Classes (also known as MFC [151]) provided by Microsoft as part of its Visual C++ product [150] are examples of comprehensive collections of interrelated classes, providing a variety of components and mechanisms that simplify the development of applications using rich graphical user interfaces.

A fundamental difference between PowerPlant and MFC is that while PowerPlant is a forest of classes, which encourages the judicious use of the — sometimes controversial, as discussed by Cargill [136] — C++ multiple inheritance mechanism, MFC is virtually a single tree, and has very few disconnected utility classes.

Although debates over which approach is "better" have, over time, reached the status of religious wars between various professionals, the usefulness and applicability of application development *frameworks* has only expanded, regardless of their fundamental design approach, as shown by the emergence of new *frameworks*, such as Cocoa [155] for MacOS X [160].

4.5 Summary

In this chapter, we have examined the software engineering context of our research. We have explored the core human interface design principles that needed to be taken in consideration, and we have shown their implications for our work.

We also briefly outlined the challenges of image and speech compression, and introduced our chosen compression technologies that proved to best fit our requirements, PNG [218] and GSM 06.10 [227].

We have then presented the concept of *design patterns* [127] and described a selection of such patterns that have applicability in the AudioGraph system context. We have also outlined the role that application development frameworks play in modern software construction.

In the next chapter, we will take a more in-depth look at the software architecture of the AudioGraph system, and examine working examples of the design patterns that permeate it.

5 Software Architecture

In this chapter we present the software architecture of the solutions that the author has designed and constructed in the course of this research. We use the architectural description approach popularised by the Unified Process [146], which contains a rich collection of software development best practices, and is widely used in the industry.

In this context, the software architecture description captures a number of views that address the concerns of various stakeholders — such as users, analysts, designers, developers, testers or project managers — in order to provide an integrated representation of the system.

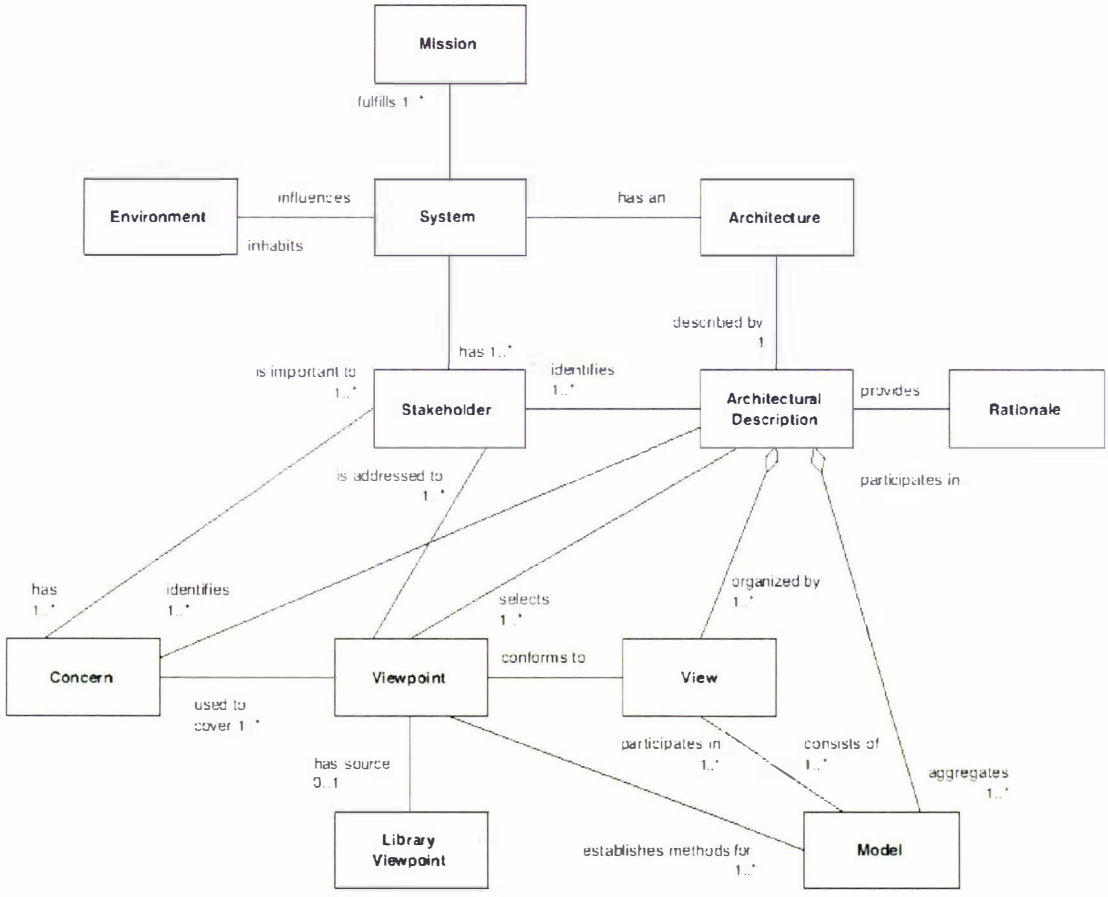


Figure 5.1 Conceptual model of architectural description (IEEE 1471)

This approach is congruent with the IEEE 1471 [139] recommended practice for architectural description of software-intensive systems. We take advantage of the elements of the conceptual model of architectural description it introduces [139] (pp. 5), shown in Figure 5.1, to emphasise the high-level design decisions made and trade-offs faced by the author during the course of this research and design effort.

5.1 Mission

The principal focus of the AudioGraph system is simplicity. Its immediate aims are to quickly and effectively capture the traditional classroom lecturing experience into a form that makes easy web deployment possible.

The following sequence of events could perhaps be considered typical: the lecturer presents a slide on an overhead projector, explains a few points and makes a few annotations on the slide or on a whiteboard perhaps, and then moves on to the next slide, where the explanation and annotation cycle repeats. It is a simple, predictable, clear sequence of actions.

Using a recording application, lecturers prepare educational material as if they were delivering it to a classroom. They prepare slides and they make graphical and voice annotations, just as they would do in a classroom setting. When the authors are satisfied with the recorded material, they can export it into a special format, optimised for web-delivery.

Once the presentation is available in this format, students may then view the recorded material at their convenience, using either a standalone player application, or a web-browser with a suitable plug-in that can render AudioGraphic lectures.

5.2 Architectural Representation

In summary, lectures are captured with a recorder application, and presented with a browser plug-in or a standalone player. The following views, addressing mainly the concerns of interest to designers, analysts and developers, are captured for these applications in the coming sections:

- Use case view — outlining the main use cases.
- Logical view — providing an overview of the structure of the applications and introducing the main underlying components.
- Deployment view — clarifying the distribution of the components in the system.

Performance and quality issues are also appropriately highlighted where they are relevant.

In terms of notation, all class names beginning with `L` and most class names beginning with `U` represent PowerPlant [148] library and utility classes respectively, while class names beginning with `C` represent custom classes created by the author. In the various UML diagrams [145], abstract element names — classes and methods — are shown in *italics*.

5.3 Architectural Goals and Constraints

In this section, we describe the main software requirements, objectives and constraints that have a significant impact on the architecture. For instance, in the pursuit of simplicity in educational

content creation, we aim to present users with a simple interface, based on a sequential playback model, rather than adopting more complex, hierarchical structures, or with potential provisions for concurrency.

In order to reduce the development effort required, and given our initial focus on the Macintosh platform, we have used the PowerPlant [148] application development framework, which provided a wide array of application and user interface building blocks.

In terms of output for the web-ready presentations, we have aimed to use only PNG [218] and GSM 06.10 [227], which have been described in Sections 4.2.1.1 and 4.2.2.1 respectively.

However, support for multiple input image and sound data formats while recording was also considered most important, in order to provide users with flexible authoring options. To address this requirement, the QuickTime [228] technology was used in order to gain access to a large collection of codecs.

In the following sections, we will briefly introduce the PowerPlant [148] framework and the QuickTime [228] technology, which have also played important roles in the technical realisation of our system.

5.3.1 PowerPlant Framework

Two application development frameworks have gained significant market share on their respective platforms: PowerPlant [148] is targeted towards the Macintosh platform and MFC [151] towards the Windows platform.

There is a fundamental design difference between the two frameworks. MFC [151] adopts a *tree* hierarchy approach. In other words, virtually all classes in MFC [151] inherit from a single ancestor, `CObject`, and multiple-inheritance is hardly ever used. The emphasis in MFC [151] is placed on using the *Class Wizard* functionality to define new functionality by overriding inherited methods or providing message handlers at different levels in the structural hierarchy.

In contrast, PowerPlant [148] emphasises the judicious use of multiple-inheritance and adopts a *forest* approach, defining clusters of classes grouped by classes of functionality. PowerPlant [148] encourages users to mix and match functionality as required through the careful use of multiple-inheritance.

Both frameworks support the document-view paradigm. In other words, both frameworks provide base classes for applications, documents and windows that provide views of documents, and both frameworks make extensive use of the *template method* design pattern [127] (pp. 325) in implementing this infrastructure.

Both frameworks provide mechanisms for creating windows and their associated controls based on definitions stored in resource files. Metrowerks provides the *Constructor* application as part of its CodeWarrior suite [147] for editing resource files and defining a precise mapping between resources and the appropriate C++ classes. A variant of the *memento* design pattern [127] (pp. 283) is used to implement this capability.

PowerPlant [148] also provides a wide variety of utility classes and design pattern realisations that can be put to good use in developing robust and efficient applications. Various examples of the application of such mechanisms are outlined throughout this chapter.

5.3.2 QuickTime Technology

QuickTime [229] is a collection of services, with application programming interfaces in multiple programming languages. Applications may make use of QuickTime [228] to manipulate and control time-based data structures, called *movies*. Using QuickTime, applications can manipulate movie data by creating and editing it using the familiar paradigms of text and still-image graphics editing [229].

In addition to processing video data in a wide variety of formats, QuickTime can handle multiple sound channels encoded in various formats, MIDI music, vector graphics, sprites, 3D objects, virtual reality panoramas and objects, hyperlinks, and text for captioning or other purposes [228]. QuickTime can be used to add a very wide range of media-rich features quickly and cost-effectively to most applications.

The QuickTime [228] technology is supported by both the Macintosh and Windows platforms. Although from a technical perspective the MacOS [159] and Windows implementations of QuickTime [232] are structured differently, the application-programming interface presented to applications on both platforms is virtually identical. All QuickTime capabilities [229] are available on both platforms. On the Macintosh platform, QuickTime [228] is implemented as a set of system extensions, while on the Windows platform it is implemented as a dynamically linked library [152].

This technology is widely available, and it can be licensed from Apple for inclusion in software or hardware products. The required components can be downloaded form the Apple site [232], for both the Macintosh and the Windows platforms. Many computer hardware suppliers ship the QuickTime [229] technology as built-in software.

The latest version of QuickTime [229] — QuickTime 6 — supports over 200 media components and capabilities, including most common multimedia and compression standards. QuickTime has an open architecture, so that future new media formats will be automatically supported by applications currently developed against the QuickTime API.

At its core, QuickTime [229] is a patented, extensible track-based file format. Each track may carry a different kind of content payload, such as video, audio, HTML [191] content, interactive content — such as Flash [88] and SMIL [230] — and much more. As new technologies emerge in the digital media space, new track types may be developed to support them. Appendix B offers a closer look at the core elements of the QuickTime technology [228].

This capability has been proved repeatedly during the course of time: media created with the first version of QuickTime [229] in 1991 can still be played back using current players. In addition, this also means that authors can serve multiple distribution methods with a single file, e.g. streaming over the web, downloading from a web server or local playback from a CD for both Windows and Mac users.

In the AudioGraph system, we use QuickTime [229] services, if they are available, for playing sounds and rendering images. QuickTime [229] technology may also be used for image transcoding, when it is necessary to convert efficiently between image formats. For instance, the AudioGraph recorder currently transforms all images in a lecture to the PNG [220] format when exporting to the web-ready file format.

5.4 Recorder Architecture

The recorder is the application used to create, edit and export lecture material in the web-ready file format. On the Macintosh platform, it relies on the PowerPlant framework [148].

5.4.1 Use-Case View

The recorder application is focused on importing, creating, editing and exporting multimedia lectures. The use-case diagram in Figure 5.2 applies to all versions of the AudioGraph recorder from 1.0 onwards.

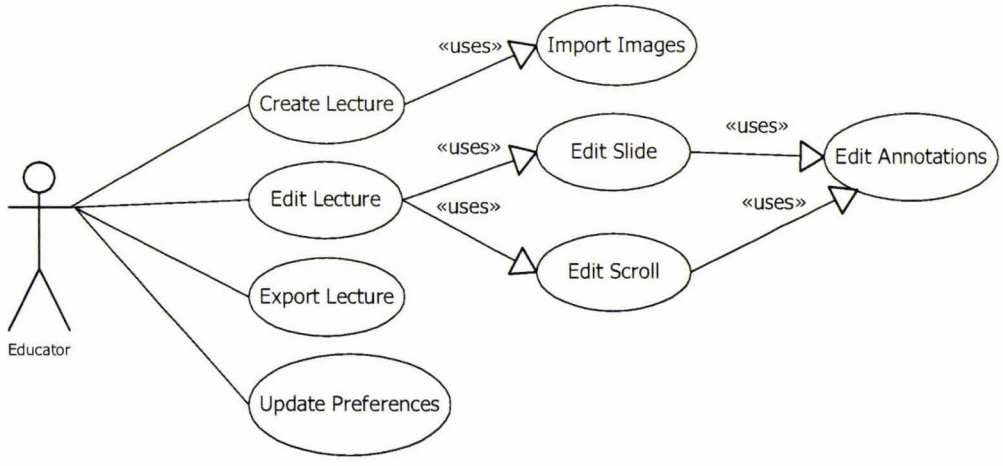


Figure 5.2 Recorder high-level use case diagram

Educators are the main users of the AudioGraph lecture recorder. They may create lectures, either from scratch, or by importing a collection of images — which may be slides from existing material created with presentation software.

When editing lectures, users may either edit slides or scrolls, and in the process they would typically manipulate annotations as well, such as voice, image or graphical annotations.

Users may also export lectures in the AudioGraph web-ready file format, thus making them available for web-delivery.

Finally, users may update application preferences, controlling sound, graphical, editing and exporting details. Section 3.5 provided more details associated with these use cases.

5.4.2 Logical View

The main task of the AudioGraph recorder application is to keep track of *lecture* documents. It is also responsible for providing the command-handling framework, and maintaining a collection of preferences that affect various tools and options available to the author. Figure 5.3 shows the AudioGraph application's class diagram.

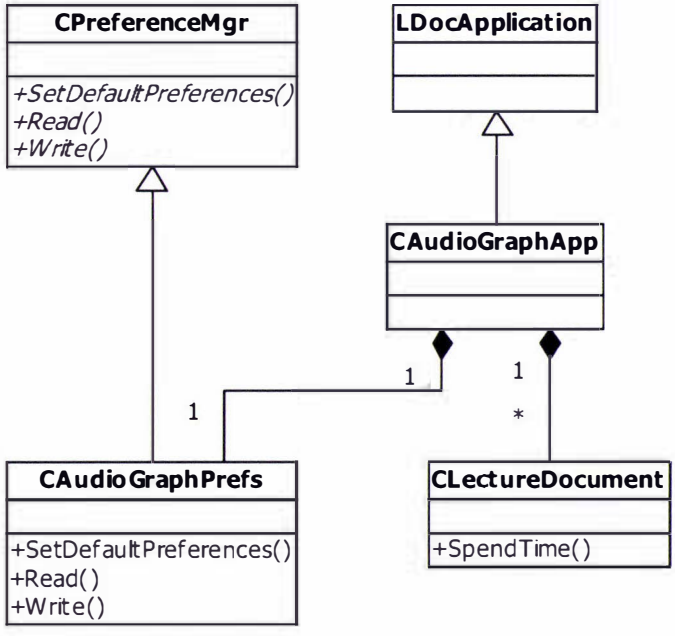


Figure 5.3. Recorder application class diagram

`LDocApplication` is a skeleton application class provided by the PowerPlant [148] framework, which offers basic scaffolding for managing documents.

The application (`CAudioGraphApp`) sets up the various menus that facilitate file manipulation — opening, closing saving and exporting lecture *document* files — editing — cutting, copying, pasting various elements in lecture and episode windows — and windowing

commands — showing and hiding various palette windows, switching among the various open windows, etc.

Using the *command* [127] (pp. 233) and the *chain of responsibility* [127] (pp. 223) design patterns, menu item selections are transformed into actions and dispatched to the most appropriate object instances for execution, and undo/redo functionality is provided for a subset of them as well.

Application preferences (**CAudioGraphPrefs**) are stored in the *Preferences* system folder, in accordance with the Apple Human Interface Guidelines [158]. The preferences file is organised as a sequence of bit-packing structures that reflect the various categories of preferences available, as described in Section 3.5.3. The **CPreferenceMgr** is a re-useable preferences handling class, which handles the details of storing and retrieving preferences data blocks to and from the appropriate locations, depending on the operating system version.

CAudioGraphPrefs takes advantage of the **CPreferenceMgr** capabilities and makes available the preference category structures — sound in, graphics, editing and general, as seen in Figure 3.13 through to Figure 3.16 — to the application class, which exposes them to clients as static member functions. The various attributes available in these preference category structures are then made available directly to client sections of code, such as:

```
CAudioGraphApp::GetSoundPreferences().compressionType = kNoCompression;
```

In retrospect, the preference category structures could have hidden the data values and provided specific access methods, to effectively hide implementation details and provide more robustness. For instance, the preferred sound sample size is being stored as a **Fixed** value, which has a special binary representation, as required by the Macintosh Sound Manager [157]. Clients of the preferences structure need not know this special requirement if appropriate access mechanisms are provided.

The data for the various preference categories is synchronised with the preference file as a block of data rather than individual values. The application also has the ability to restore the user preferences to a recommended set of values, called *factory settings*.

If the version of the recorder application stored in the preferences file does not match the version of the application being run, then the preferences file is discarded and the *factory settings* are used instead, which are a recommended set of values. This mechanism is useful when the structure of the preferences file changes in subsequent versions.

In this context, the alternative of providing transition paths for preferences coming from earlier versions of the recorder would require maintaining knowledge of the various structures as released with the different versions of the recorder. Clearly, this approach could rapidly become

cumbersome, particularly if the preferences structures are likely to change noticeably from one released version to another, as has been the case with our rapid iterations.

A potential alternative that could mitigate this drawback would be to use tagged values for the preference attributes, perhaps represented as an XML [169] document. Using this approach, the various supported preference items could be migrated to new versions, the ones no longer supported would be discarded, and any new preference items would be set to their factory settings default values. Any semantic conflicts between different versions can be resolved in a standardised manner by making use of the XML namespace mechanism [170].

Further on in this section, we will examine the core underlying components and mechanisms of the recorder application:

- Lecture Document — the main documents edited by the recorder.
- Episodes — the building blocks of lecture documents.
- Views — the constructs that render episodes and lectures and their associated controls.
- Authoring Components — the annotations contained in episodes.
- Data Transfer — the mechanisms for loading and storing lecture documents and preparing web-ready files and websites.

5.4.2.1 Lecture Document

The AudioGraph recorder application edits *lecture* documents. Each lecture document may contain a collection of episodes. Lecture documents can also be exported into the web-ready file format, either as *compact* presentations or as *expanded* websites, as discussed in Section 3.5.1. Figure 5.4 illustrates the lecture document's class diagram.

The *lecture* document (**`CLectureDocument`**) uses a special window class (**`CLectureWindow`**) to present list or thumbnail views of the contained episodes and provide commands for creating and editing them, as shown in Figure 3.5 and Figure 3.6. **`LDocument`** is a PowerPlant [148] skeleton class for document management.

Most modern applications that interact with users are driven by messages processed through an event loop. On a Macintosh [159] system, when the operating system is otherwise unoccupied, *idle* events are passed through the event loops.

The **`LPeriodical`** class provided by the PowerPlant [148] framework and the associated idiom exploits these facts, by providing the ability to execute code either as each event is just about to be processed by the system — by invoking *repeaters* — or as each *idle* event is ready to be processed — by executing code supplied by *idlers*. **`LPeriodical`** is an abstract base

class that requires its descendants to implement a **SpendTime()** method that would perform the periodical computation.

The **LPeriodical** class is typically used as a *mixin* class, being used in conjunction with other base classes in multiple-inheritance structures. For instance, the **CLectureDocument** derives from **LDocument** and **LPeriodical**, as shown in Figure 5.4.

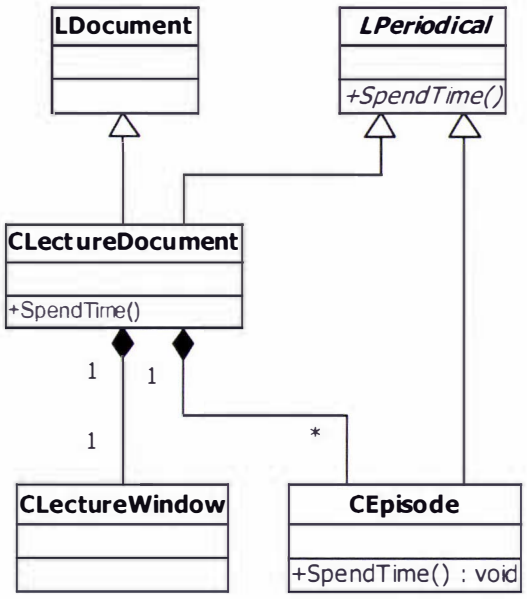


Figure 5.4. Lecture document class diagram

The **LPeriodical** class maintains two protected static lists for active *repeaters* and *idlers*. The framework calls the following two static methods:

```
LPeriodical::DevoteTimeToRepeaters();
LPeriodical::DevoteTimeToIdlers();
```

In the appropriate locations in the event loop in order to trigger the execution of **SpendTime()** for all the currently active *repeaters* and *idlers*, implementing a variant of the *observer* design pattern [127] (pp. 293). The same mechanism is available in the web browser plug-in framework as well, even though the event processing structure is significantly different compared to an application.

The ability to perform tasks at times when the system is otherwise free is very convenient, and we have put it to good use in a number of circumstances.

For instance, the lecture window displays estimated episode durations and the aggregated estimated lecture duration. To calculate these values in an unobtrusive way, we transformed the lecture document into an *idler* by having it inherit from the **LPeriodical** class.

The **SpendTime()** method that implements the lecture document *idler* checks whether any episodes in the lecture have changed at all. If so, it stops idling itself and starts the episode in

question idling in order to determine the correct episode duration. Once the episode finishes this calculation in due course, it restarts the lecture document *idler*, which adds the current value to the total and updates the appropriate visual elements in the lecture window. The process continues indefinitely, maintaining the episode and lecture durations up-to-date and using only spare computing capacity.

The document class is responsible for triggering the opening, saving and exporting of files. It implements methods that are called by the PowerPlant framework [148] in response to user commands such as *Open*, *Save* and *Save As...*. Once the lecture document changes from its previously saved state, it may also be reverted to that state by invoking the *Revert* command.

The lecture document also implements the functionality required for adding new episodes and inserting them at the appropriate location in the sequence of episodes it maintains.

5.4.2.2 Episodes

An episode contains a collection of annotations that are recorded and played back in a simple linear sequence. We have deliberately made no provision for parallel execution (such as the **`par`** construct in SMIL [188]) in order to keep the recording experience as simple as possible. Figure 5.5 shows the episode class diagram.

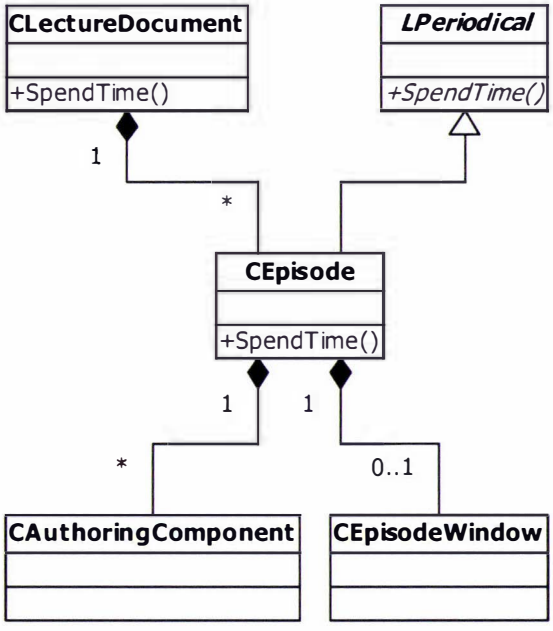


Figure 5.5. Episode class diagram

Episodes (**`CEpisode`**) are managed as a collection in the lecture document (**`CLectureDocument`**), as also shown in Figure 5.4. Episodes may contain any number of annotations derived from **`CAuthoringComponent`**. In many ways, this is a central class in the AudioGraph system, as it models the archetype for all annotations available to the author.

This class is described in more detail in Section 5.4.2.4. Each episode may also have an associated window (**CEpisodeWindow**) to display or playback its annotations.

The episode class (**CEpisode**) is also an *idler* — as described in Section 5.4.2.1 — and this functionality is used to estimate the playback duration. For these calculations, we have assumed that all graphical annotations take one tenth of a second to render. This value was chosen as a rough estimate, as the episode durations were meant to be indicative rather than extremely accurate.

Seeking near-perfect accuracy for these duration values could prove extremely difficult, as the playback-speed is affected not only by the technical specifications of the hardware on which the presentation is to be ultimately played, but also by the number and mix of applications that are running concurrently and their particular workload at the time. We must keep in mind that most modern microprocessors have non-deterministic execution characteristics due to the widespread use of caching and pipelining techniques.

Sound and pause annotations provide their own specific duration details for the episode duration estimate. Since these types of annotations are typically occupying the bulk of the presentation time, the overall estimates obtained prove to be relatively accurate and quite useful in practice.

The episode class also keeps track of the current selection of annotations. Annotations can be selected through the edit console, the edit window, or the edit menu — using the *Select All* command. All these methods of selection operate on a **CComponentCollection** object hosted by the episode, providing a host of services.

For instance, this class maintains information about the number and types of annotations in the selection, e.g. whether any annotations are of a graphical nature, so that controls and menus can be adjusted accordingly. In this particular case, the colour and pen size controls would become available, and any new colour choice would affect all graphical components in the selection. If the selection contains only audio annotations, such as sounds and pauses, then the colour and pen size controls are disabled, since they have no effect on such annotations.

If the selection contains a single item, then the *Edit* menu items and the attributes palette section titles are adjusted to indicate the particular type of component, as shown in Figure 4.2. If the selection contains multiple items, then the edit menu items are adjusted refer to the selection as a whole, giving user a clear indication of the effects the commands would have.

The **CComponentCollection** class — that is actually an extension of an array of **CAuthoringComponent** pointers — also aggregates visual regions that are useful when dragging the selection, since an adequate outline must be rendered to provide visual feedback.

Making use of the component collection, the episode forwards the appropriate commands to the annotations in the current selection, as dictated by the menu item choices made by the author. In retrospect, the responsibilities between the episode and the selection class could have been divided more in favour of the selection class, resulting in simpler code, more cohesion and marginally better performance. In such an arrangement, commands could have been dispatched directly to the selection object, and selection-processing responsibilities would have been centralised. Any callbacks required on the episode object itself could be handled elegantly, safely and efficiently by making use of *generalised functors*, as suggested by Andrei Alexandrescu [144] in his recent book, *Modern C++ Design*.

The episode class keeps track of the visibility status and locations of the various windows associated with the episode, such as the episode window, the edit console, the tools palette and the attributes window. This facility is useful for restoring the editing context of individual episodes to the last state in which the author has used them, which we assume to be the preferred layout.

Users can determine whether they wish to make use of this capability by adjusting a preference item on the *Editing* preferences panel, as described in Section 3.5.3.3. If not, then the system calculates recommended window sizes and positions according to the current display settings.

5.4.2.3 Views

A number of distinct windows and views are available to facilitate interaction with the AudioGraph recorder application.

The PowerPlant [148] framework provides pane, view and window classes to facilitate modelling user interfaces. **`LWindow`** derives from **`LView`**, which derives from **`LPane`**, as shown in Figure 5.23.

Panes are the basic visual elements, and they are responsible for managing size and position information. Views are panes that may contain other panes, and are responsible for propagating commands to all children panes. Panes cannot host other panes. In effect, this is another example of the *composite* design pattern [127] (pp. 163) at work.

In this context, views may represent visual areas in a 32-bit coordinate domain. Considering the normal resolution on a Macintosh computer is 72 pixels per inch, views might cover square areas with nearly five million feet on a side, which for most practical applications is much more than enough.

Windows are views that also correspond to a native Macintosh window, linking the abstract PowerPlant [148] visual class hierarchy to the concrete operating system windowing details.

5.4.2.3.1 The Lecture Window

The lecture window (**`CLectureWindow`**) gives an overview of the lecture document. It can present the collection of episodes either as a list of episode titles, or as a list of episode miniature representations (thumbnails), as shown in Figure 3.5 and Figure 3.6. Figure 5.6 represents the lecture window class diagram.

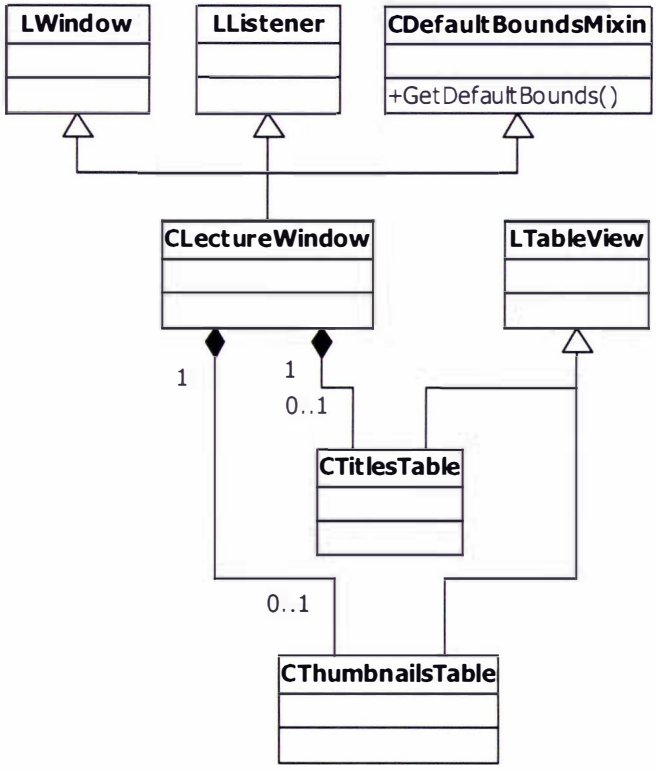


Figure 5.6. Lecture window class diagram

Details such as episode titles and estimated episode durations are presented in both views. In the titles view, depicted in Figure 3.5, appropriate icons indicate whether the episodes are slides or scrolls. You may also note that the icons also give clues about the status of the associated episode. The icons with small red marks indicate episodes whose state has been modified from its current representation on disk.

`LWindow` is the basic PowerPlant [148] skeleton class for windows. When created, the lecture window (**`CLectureWindow`**) contains at least one table, more precisely the titles table (**`CTitlesTable`**). If the thumbnails view (**`CThumbnailsTable`**) is not selected, the thumbnails table is never created, saving time and memory. If the thumbnails view were to be chosen as the default view, the situation could be reversed in its favour. Further views could be added to the lecture window in the same spirit, such as a *links* table that visualises any hyperlinks among episodes. Version 2.0 of the AudioGraph provides such a view. The table

classes (**CTitlesTable** and **CThumbnailsTable**) are derived from **LTableView**, which is a PowerPlant [148] skeleton class for tabular views.

The various buttons and other controls hosted in the window communicate with it through the *observer* design pattern [127] (pp. 293) implemented through the use of the **LBroadcaster** and **LListener** classes provided by the PowerPlant [148] framework, and which is discussed in more detail in Section 4.3.12.

The lecture window also inherits from the **CDefaultBoundsMixin** class in order provide standardised support for window-bounds persistence. Since we wished to provide window-bounds persistence for most windows in the AudioGraph recorder, it was only natural that we sought to capture the required functionality in a separate class and mix it with the appropriate window classes as required in order to facilitate disciplined reuse.

The *mix-in* idiom [127] (pp. 16) is widely encouraged by the PowerPlant [148] framework, with its focus on a family of class hierarchies in contrast to a single class hierarchy, as adopted by MFC [151]. For instance, this approach is readily apparent in the use of *periodicals* as described in Section 5.4.2.1 and in the implementation of the *observer* design pattern [127] (pp. 293) that uses the **LBroadcaster** and **LListener** *mix-in* classes.

5.4.2.3.2 The Episode Window

The episode window presents the WYSIWYG view of the episode annotations, and orchestrates the display of all the other associated control windows. The episode window class diagram is presented in Figure 5.7.

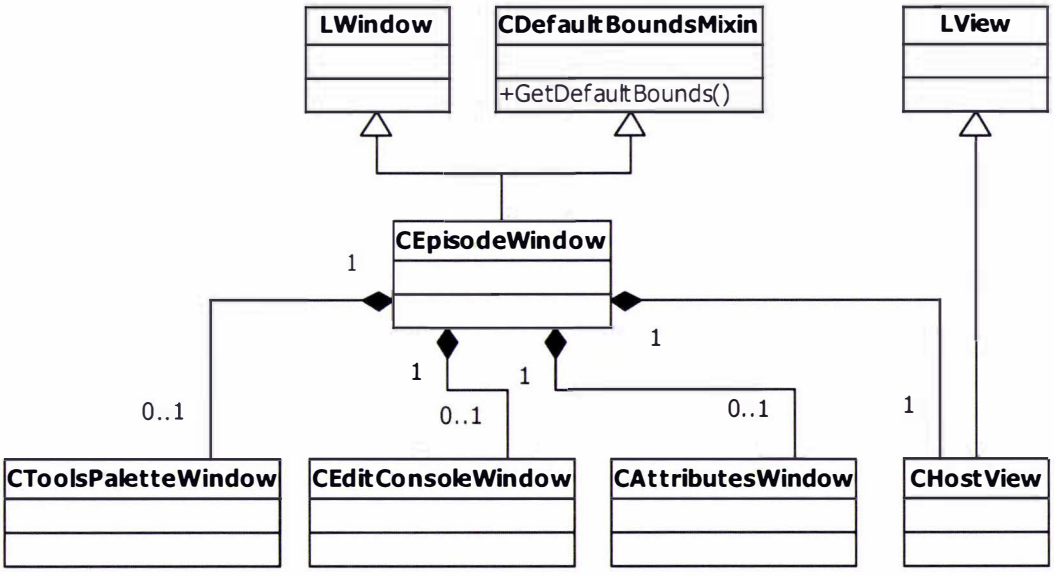


Figure 5.7. Episode window class diagram

The controls in the episode window, as shown in Figure 3.7, dictate which of the various control windows (the **CToolsPaletteWindow**, the **CEditConsoleWindow**, or perhaps the **CAttributesWindow**) is to be displayed, and whether graphical annotations should snap to the editing grid or not. Controls in the *Editing* and *General* preferences panels, as seen in Figure 3.15 and Figure 3.16, specify the default window visibility, the grid size and its status.

The episode window (**CEpisodeWindow**) does not assume responsibility directly for maintaining the visual hierarchy of the graphical annotations. Instead, it contains a **CHostView** instance dedicated to this task.

Just as the lecture window, the episode window mixes in the **CDefaultBoundsMixin** class so that it would provide the same bounds persistence capabilities. However, where the lecture document is responsible for storing the actual bounds values for the lecture window, the associated episode object stores the bounds values for the episode window and all the corresponding control windows.

5.4.2.3.3 The Host View

The *host view* contains panes for all the annotations that have visual representations. Figure 5.8 presents the *host view* class diagram.

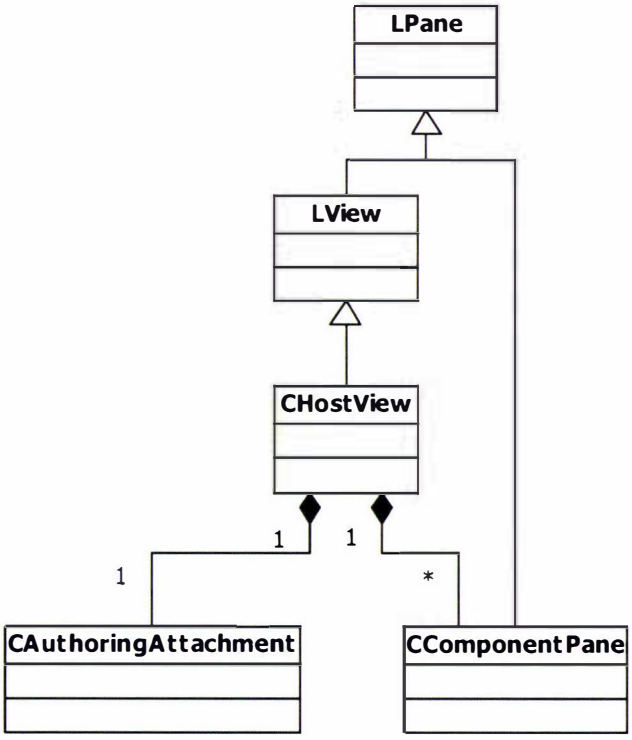


Figure 5.8. Host view class diagram

Given the native PowerPlant [148] support for creating abstract visual hierarchies, at first it seemed natural that we should adopt it for modelling the visual hierarchy of AudioGraph

annotations. However, although this approach is most convenient in terms of its ease of use from a design and annotation editing perspective, it proved to have performance limitations in the context of visual hierarchies with rapidly changing visual attributes, as is the case with the dynamic AudioGraph presentations during playback.

The main role of the *host view* class (**CHostView**) is to aggregate all the panes (**CComponentPane**) associated with the visual annotations and maintain the correct z-order information for them.

Another important use for the *host view* is to orchestrate the editing activities using an authoring *attachment* (**CAuthoringAttachment**). *Attachments*, and their role in the *decorator* design pattern [127] (pp. 175) variant in use in the PowerPlant [148] framework, are discussed in more detail in Section 4.3.10.

The authoring *attachment* can be called upon to handle a variety of messages from a number of different sources. The same *attachment* may be attached to different *attachable* hosts. For instance, the authoring *attachment* is attached to the application instance and to the *host view* instance. If the current episode is a *scroll*, then the *attachment* is also attached to the *scroll frame marker*-pane in order to make it *transparent* to click or mouse movement messages.

5.4.2.3.4 The Tools Palette Window

The *tools palette* window facilitates control over various tools and editing commands available to the author. Just as all the other main editing windows, it makes use of the bounds mix-in class (**CDefaultBoundsMixin**) in order to be capable of arranging itself where most appropriate or restoring its position where last placed by the author. Figure 5.9 shows the tools palette window class diagram.

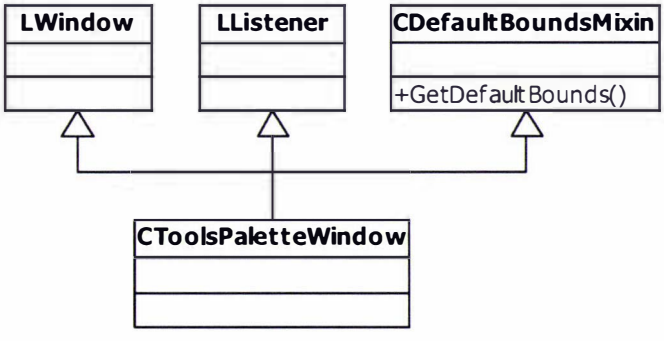


Figure 5.9. Tools palette window class diagram

The tools palette (**CToolsPaletteWindow**), besides being a window (**LWindow**), is also a listener (**LListener**), ready to handle notifications from the controls it hosts, which are broadcasters. The tools palette keeps a reference to the authoring *attachment*

(**CAuthoringAttachment**) owned by the *host view*, and in response to notification from the tool controls, it updates its state. In this manner, the authoring *attachment* keeps track of the current state of the editing tools.

As with the other windows in the AudioGraph recorder, the *tools palette* is also derived from the **CDefaultBoundsMixin** class in order to take advantage of the same bounds persistence capabilities.

When the authoring *attachment* handles click and drag messages, it makes use of the available state information in order to trigger the of the appropriate annotation-creation action. In order to support multiple undo functionality, annotations are not created directly, but rather through *actions* that can readily be undone or re-done.

Rather, creation *actions* are generated, and these may be easily undone and re-done, as required, as can be seen in this line drawing example:

```
// Create the line pane
//
    if( PointsAreDifferent( anchor, end ) ) {

        CLineComponent  *theComponent = new CLineComponent( anchor, end, this );
        ThrowIfNil_(theComponent);
        CLineAction *theAction = new CLineAction( GetWindow(), theComponent );
        ThrowIfNil_(theAction);
        GetWindow()->PostAction( theAction );
    }
```

The line *action* that is associated with the newly created line annotation is *posted* to the window, which in turn pushes it on top of the *undo* stack and executes it. If the action must be undone, then it is simply popped off the *undo* stack, undone, and pushed onto the *redo* stack.

This approach is further clarified in Section 4.3.11, where the *command* design pattern [127] (pp. 233) is explained in more detail.

5.4.2.3.5 The Edit Console Window

The *edit console* window (**CEditConsoleWindow**) facilitates the sequencing and editing of annotations. As with the other AudioGraph window classes, it uses the default bounds mix-in class (**CDefaultBoundsMixin**) for integrated window positioning services. Figure 5.10 illustrates the edit console window class diagram.

As time has passed and the software development tools (CodeWarrior [147] and PowerPlant [148]) have evolved through multiple releases, the PowerPlant [148] greyscale appearance (and hence the **GA** indicator in its name) **LGADialog** class has become deprecated in favour of the **LDialogBox** class, which has become a simplified standard covering all dialog boxes. Future versions of the AudioGraph should use **LDialogBox**, which continues to be derived from **LWindow** and **LListener** and represents a minute change to the class diagram.

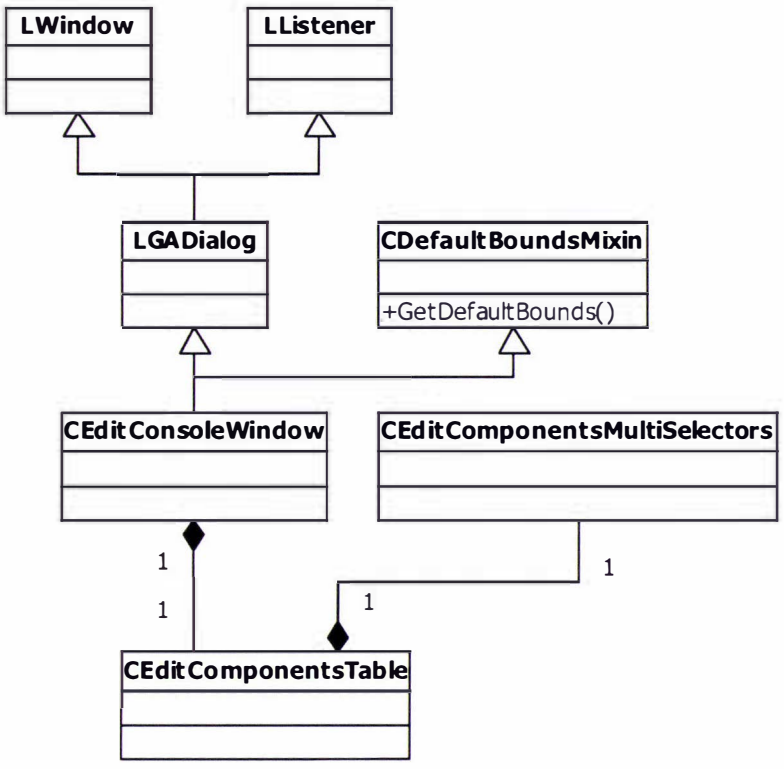


Figure 5.10. Edit console window class diagram

When the episode window (**CEpisodeWindow**) is opened, the edit console window (**CEditConsoleWindow**) is created, even if it is not required for display yet, in which case it remains hidden. The tools palette (**CToolsPaletteWindow**) and the attributes window (**CAttributesWindow**) are only created if they must be displayed.

However, the edit console window (**CEditConsoleWindow**) is created irrespective of its visual status since it contains an annotations table that is tightly integrated with the rest of the annotations infrastructure. Most importantly, this table's associated *multi-selector* class (**CEditComponentsMultiSelector**) holds the responsibility for controlling the annotation selection status. Since annotations may be selected even without any control windows being visible, the selection functionality must be available, and therefore, in the current configuration, the edit console must be available as well, even if it is hidden. Besides, episode playback controls are also tightly integrated with the edit console window (**CEditConsoleWindow**), which holds the responsibility of controlling the lifespan and activity of the **CPlayPeriodical** instance that provides playback services.

In retrospect, it would seem possible to eliminate these tight dependencies by re-factoring the responsibilities to the episode class (**CEpisode**), in order to secure a cleaner separation of responsibilities. If the episode class (**CEpisode**) were to hold responsibility for the

CPlayPeriodical instance, then playback could be performed regardless of the existence of the edit console window (**CEditConsoleWindow**).

The trouble lies in the fact that the **CPlayPeriodical** instance relies on the **CEditComponentsTable** instance provided by the edit console (**CEditConsoleWindow**) for various elements of functionality, such as visually indicating the playback status by updating the focus to the appropriate icon in the components table. To solve this issue, we could rearrange responsibilities so that the **CPlayPeriodical** instance would rely only on the collection of annotations held by the episode class, and any change in playback status would be reflected to the edit console and its components table through a variant of the *observer* design pattern [127] (pp. 293). In addition, we would also make the episode class responsible for the lifetime of the play periodical (**CPlayPeriodical**) instance, or perhaps even aggregate it as a member object.

The edit console, which could thus become optional, if it exists, would be interested in knowing which component is currently to be displayed as *selected* as playback unfolds. However, due to the use of group annotations, which may aggregate an arbitrary number of other annotations in the spirit of the *composite* design pattern [127] (pp. 163), determining the appropriate annotation to select during playback is not necessarily trivial, as potentially group membership might need to be established.

This could be accomplished either dynamically, or by storing some form of *parent* reference with each annotation. If no parent reference is maintained in the children instances, then group membership status can only be determined dynamically by recursively traversing the group hierarchy. On the other hand, if we are prepared to store a parent reference with each instance in the hierarchy, then determining the root parent annotation becomes very efficient. This situation reflects yet another trade-off between memory usage and execution performance.

To assist in the quest for a centralised playback and selection mechanism, without dependency on the edit console window, a multi-selector instance could be created in the episode class (**CEpisode**) and shared with the editing table in the edit console window as required. This course of action would lead to a solution in which the edit console (**CEditConsoleWindow**) would no longer be so tightly coupled to the rest of the system, leading to a more elegant and robust solution.

The edit console (**CEditConsoleWindow**) is also a listener (inheriting the **LListener** from its dialog ancestor), and is therefore capable of handling the notifications from the various buttons it hosts. It provides playback, positioning, information, grouping and scrolling services, as described in more detail in Section 3.5.2.2.

5.4.2.3.6 The Attributes Window

The attributes window (**CAttributesWindow**) facilitates control over various annotations attributes and provides access to advanced capabilities, like transparency control and special attributes such as *stealth*. It also uses the default bounds mix-in class (**CDefaultBoundsMixin**) for integrated window positioning services. Figure 5.11 shows the attributes window class diagram.

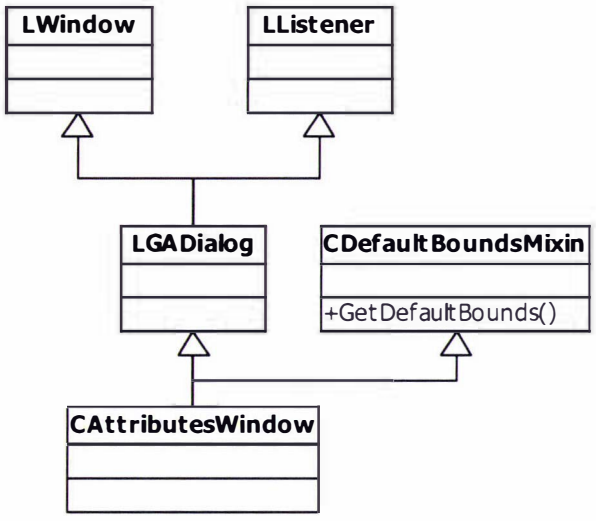


Figure 5.11. Attributes window class diagram

In similar fashion to the edit console window (**CEditConsoleWindow**), future versions of the AudioGraph should use **LDialogBox** rather than **LGADialog**.

As can be seen in Figure 3.12, the attributes window (**CAttributesWindow**) provides a progressive disclosure mechanism. This is a desirable attribute of graphical user interfaces, also recommended by the Apple Human Interface Guidelines [158], which is intended to present users with as little complexity as possible. As users become more adept at using the application, they can then gradually access the more advanced features.

In our specific case, by default, only the colour and pen size controls are available. They are laid out in a way that would be easy to use for authors wielding a pen input device, similar in intent to a painter's colour palette. Simple taps with the pen over the controls would make pen size and colour selection very easy and convenient.

Progressive disclosure is noticeable on another level as well. In the very beginning, when this project was launched in 1997, screen real estate was relatively limited, and many users were still limited to very small screen resolutions such as 640 by 480 pixels. In such a situation, there would hardly be any room for the attributes palette on the screen if we were to have a reasonably sized episode window (**CEpisodeWindow**).

Therefore, we set the recommended default (factory) settings to suggest that the attributes window should not be displayed when an episode window (**`CEpisodeWindow`**) is opened, as the frequently used colour and pen size selection-functions can be accessed through the tools palette window (**`CToolsPaletteWindow`**), albeit requiring two clicks for the menu selection process rather than a single click as in the attributes window (**`CAttributesWindow`**).

The advanced controls section of the window facilitates precise control over special characteristics of the visual annotations.

The transparency of graphical annotations can be adjusted continuously using the slider control, covering the whole range of 0 to 255 for transparency values, from opaque to fully transparent — transparency information is stored as a byte, leading to 256 possible values. Due to restrictions on screen real estate, the slider is only wide enough to actually allow the selection of just over 200 distinct values rather than the full 256. However, this level of precision is sufficient for the vast majority of users.

Perhaps in the not too distant future, as technology has advanced and display prices have dropped substantially, as more and more users will be shifting to higher resolution screens of 1024 by 768 pixels and beyond, it would be possible to widen the attributes palette slightly by 20% or so in order to cover the full spectrum of 256 transparency values.

Graphical annotations may be marked as erasing or highlighting, but not both, as their effects are mutually exclusive. However, the *stealth* and *fill* attributes are cumulative. The fill attribute only affects rectangle, oval and arc annotations.

It is conceivable that the *fill* attribute could be extended to free-hand pen annotations. In this case, setting the filled attribute would close the annotation poly-line by joining the end point with the start point and filling the resulting polygons.

The *stealth* attribute may affect any graphical annotation. Whereas normal graphical annotations are specified in the web file format definition to be followed by an implicit pause of controllable duration, which would trigger a display update, stealthy annotations are to be played without pause between them, and therefore without any intermediate display updates. As a result, stealthy annotations are very useful in animations involving large numbers of separate components.

5.4.2.3.7 The Preferences Window

The preferences window (**`CPreferencesWindow`**) facilitates control the over various categories of application preferences, as described in Section 3.5.3. As the preferences window is most likely to be used rather infrequently, managing its bounds details as done for the editing

windows was not considered necessary. Figure 5.12 gives the preferences window class diagram.

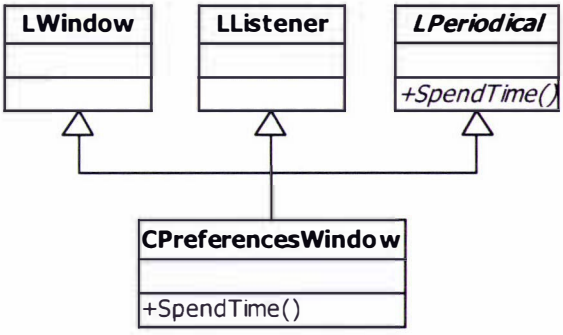


Figure 5.12. Preferences window class diagram

It is interesting to note that the preferences window (**CPreferencesWindow**), besides inheriting from **LWindow** and **LListener**, is also a periodical (**LPeriodical**). This capability is used as a *repeater* in this context to provide validation services for the various controls available.

```
void
CPreferencesWindow::SpendTime(const EventRecord &inEventRecord) {

    mReportInvalidStuff =  mReportDelay > kAlarmDelay;
    switch(inEventRecord.what) {

        case nullEvent:
        case keyDown:
        case keyUp:
        case autoKey:
            CheckControlsChanged();
            break;
        default:
            ValidateControls();
            break;
    }
}
```

The above code segment illustrates the approach we have taken. As modifications are being made, the state of the *Save*, *Revert Panel* and *Factory Settings* buttons can be updated accordingly, as shown in Figure 3.13. If it is determined within one second that the values recently entered by the user are inconsistent, then a suitable warning dialog is presented and the offending controls are restored to the last available valid values.

However, as soon as the preferences window is deactivated, the *repeater* is disabled, and only re-enabled when the window is activated again. When used as a *repeater*, the **SpendTime()** periodical pure virtual method implemented by the window is invoked as each event is processed by the application, giving tighter control than a simple *idler* could.

In addition, once created, the preferences window is not destroyed until the application quits. When closed, the preferences window merely hides itself until it is required again.

5.4.2.4 Authoring Components

In the source code, all annotations are referred to as *components*. At the root of the component class hierarchy is the **CAuthoringComponent** class, responsible for managing basic attributes and providing an abstract interface that all components must implement in order to integrate seamlessly within the rendering, editing and playback frameworks.

The authoring component inherits two other classes, the **CComponentFlags** and **CDescriptionMixin**. The component flags class provides suitable access methods for all annotation attributes, which are stored in a compact 32-bit variable. Although this implementation may be considered equivalent to a structure containing bit fields, the approach we have taken makes it easier to extend the attribute collection safely, while allowing arbitrary choice of position for the flag bits. To achieve a similar effect with a bit-field structure, we would have had to juggle various *filler* dummy fields.

The description mix-in class facilitates the association of descriptions with any annotation. Using this mechanism, descriptions can also be associated with any other classes, such as the **CResource** class for instance, in a uniform manner without having to re-implement the same functionality multiple times in the application. This is an example of a useful re-factoring [205] device. A re-factoring catalogue has been established [206] in an effort to establish best practices for improving the design of existing code.

5.4.2.4.1 Visual Components

Annotations that have visual representations are all derived from **CVisualComponent**. This class provides unified methods for visual region management, drawing infrastructure, attribute change management and movement. Figure 5.13 introduces the class diagram of the visual component class hierarchy.

All visual annotations have an associated visual region, which maps to a native Macintosh **Region**. This region is very useful to promote homogenous handling of annotations in display and selection. Actually, for rendering purposes, the component pane (**CComponentPane**) does not make any distinction between the pen, line, rectangle, oval, and arc components. Only annotation attributes affect the ultimate rendering effect, as dictated by their specific values.

However, picture annotations (**CPictureComponent**), and their associated vanishing image annotations are handled differently, since their effect must be manifested on the background layer. Even so, it is not out of the question that the file format might be extended to allow picture annotations to have either background or foreground effect.

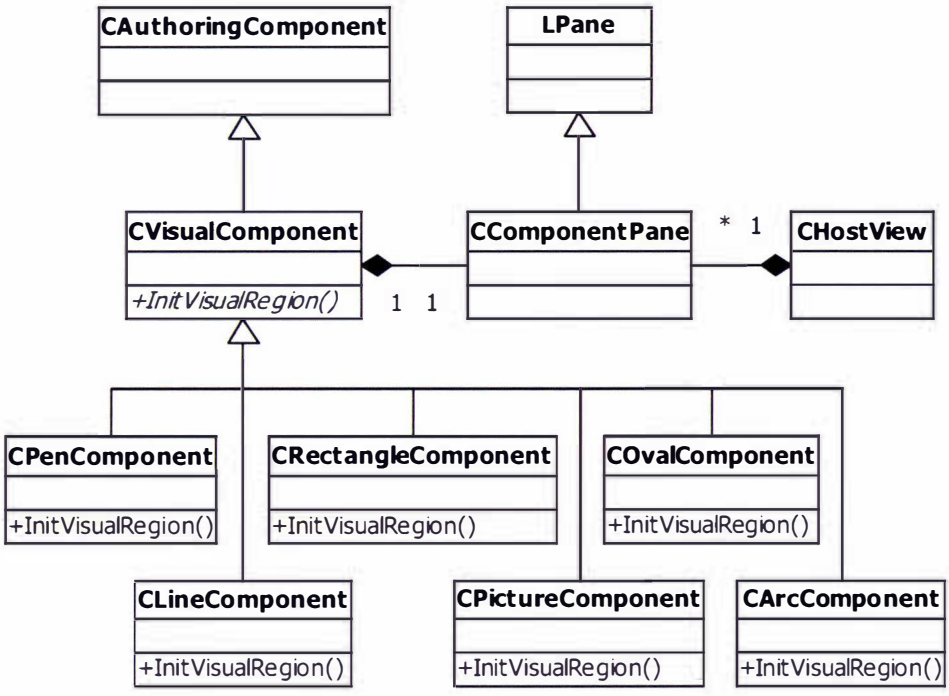


Figure 5.13. Visual component class hierarchy

With the advent of text annotations in version 2.0 of the recorder, which are implemented using highly optimised PNG [218] images in order to preserve the exact experience during playback without requiring the matching font to reside on the target platform may be seen as an excellent opportunity for the introduction of foreground images. As a significant advantage, if images were displayed in the foreground, their effect could be erased very easily — and selectively — using a simple erasing annotation, such as an erasing filled rectangle for instance, rather than using a more complex image vanishing annotation which would cause the whole image to disappear. In terms of processing complexity, it is much more efficient to erase foreground annotations than to *vanish* pictures.

5.4.2.4.2 Audio Components

The sound and pause annotations provide the soundtrack of the AudioGraph presentations. Figure 5.14 illustrates the class diagram of the audio components class hierarchy.

Sounds can be recorded either directly, by pressing the sound recording tool in the tools window, or by using a dedicated sound editing dialog that also facilitates editing sound transcripts, as shown in Figure 4.4. To accommodate these requirements, the sound recording functionality has been extracted into a separate class, the `CSoundPeriodical`.

The periodical aspect of this class — used as an *idler* in this instance — is intended to monitor the sound recording progress and provide dynamic updates on the sound input level, elapsed time and input gain settings to any interested parties. The notifications are provided through the

observer design pattern [127] (pp. 293), as implemented in PowerPlant [148] using the **LBroadcaster** and **LListener** classes. Specifically, sound recording progress dialogs are listeners and update the states of the appropriate controls as required, as seen in Figure 4.4.

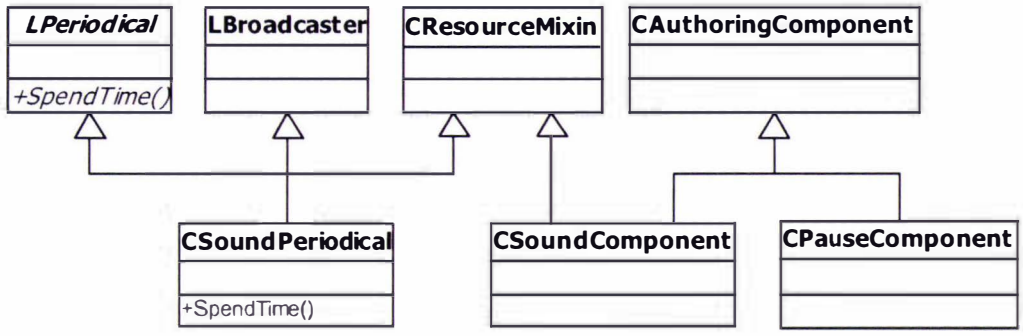


Figure 5.14. Audio components class hierarchy

In order to store the sound data, the sound periodical makes use of the capabilities provided by the **CResourceMixin** class. This class provides a suitable abstraction for dealing with resources, freeing client code from the necessity of dealing with the Macintosh Resource Manager directly. Using this approach, we can make use of the optimised data management capabilities of the Resource Manager for storing our digital sound samples.

Sound annotations (**CSoundComponent**) also take advantage of the same resource mix-in class, maximizing code reuse and providing increased robustness. For sound playback, the **CAsynchronousSound** class is used, and it extracts sound data from the specified resource.

The asynchronous sound class is also a broadcasting periodical (inheriting from **LBroadcaster** and **LPeriodical**), also used as an *idler*, just like the sound periodical. However, the asynchronous sound class is used as a *singleton* [127] (pp. 127), ensuring that only one sound is played at any one time. In retrospect, this restriction could be relaxed, and multiple presentations could be allowed to playback simultaneously, using the sound mixing capabilities provided transparently by the QuickTime [229] technology.

In the web file-format specification, pause annotations (**CPauseComponent**) are defined with a millisecond resolution. However, the AudioGraph recorder only captures pause durations in increments of a tenth of a second — one hundred milliseconds — as it was judged accurate enough for most practical uses. For ease of implementation, since the earlier MacOS platforms [159] offered a simple time delay mechanism with a resolution of 1/60 second, called a *tick*, pause durations are internally converted into the appropriate number of *ticks* and delays are measured in this unit.

With the advent of the MacOS X [160], which is based on a UNIX [207] core and is a true pre-emptive multi-tasking operating system, advanced time services have become easily available.

As a result, it is likely that the implementation of pause intervals will be adapted to use millisecond resolution directly, as the *ticks* interface is deprecated.

5.4.2.4.3 Other Components

Besides the visual and audio components, the AudioGraph model makes use of a number of other annotations. Figure 5.15 shows the class diagram for these.

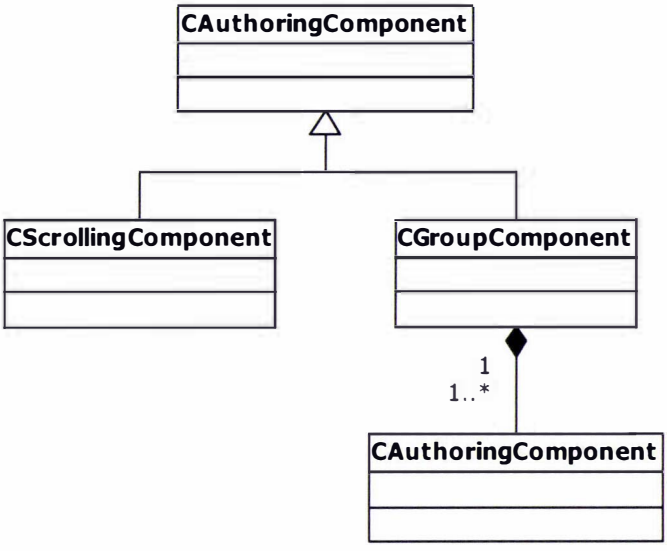


Figure 5.15. Class hierarchy for the other annotation components

The scrolling annotations (`CScrollingComponent`) move the origin of the host view (`CHostView`) inside the image coordinate space. Currently, the movement is only vertical, simulating the effect of the typical transparency scroll. However, many whiteboards feature horizontal scrolling rather than vertical. In the future, perhaps the scrolling movement capabilities could be generalised to both the horizontal and vertical directions.

The group annotations (`CGroupComponent`) are only provided for editing convenience in the recorder application. Group annotations are a realisation of the *composite* design pattern [127] (pp. 163) in this context, facilitation the aggregation of annotations in collections that can be treated as individual annotations in their own right. This mechanism is particularly useful for encapsulating animations consisting of long sequences of stealthy pen annotations, allowing the author to focus more easily on the main elements of her presentation. The group annotations have no counterpart in the web-ready AudioGraph files, as they add no semantic value to the presentation, being merely an authoring aid.

Conversely, as later versions of the AudioGraph recorder have started to make use of new kinds of annotations, such as *halt*, *clearscreen* and *hyperlink* annotations, they are supported by the version 2.0 of the AudioGraph web-browser plug-ins.

It is interesting to note that the *halt* annotations affect the playback experience not by their behaviour, but by their type. In other words, the *halt* annotation has no functionality directly associated with it. Instead, the playback loop determines if the type of the current annotation is that of a *halt* component and if so takes appropriate action by stopping the presentation.

Clearscreen annotations, besides having the obvious effect of clearing the screen of all foreground and background graphics and establishing a background colour, have an interesting capability: after a *clearscreen* annotation has been displayed, in the context of linked episodes, all images that have been indicated as *persistent*, will continue to be displayed. This behaviour applies even for self-linking (never-ending) presentations.

Another interesting convention used in version 2.0 of the AudioGraph web-ready file format specification is that if presentations do not contain a *clearscreen* as their first annotation, the player should generate for them a synthetic *clearscreen* annotation. This ensures that each subsequent episode linked-to in the same player instance is presented in its own context, unless the author specifically chose to mark a number of *persistent* images in order to have them continue to be displayed in the new episodes.

5.4.2.5 Data Transfer

AudioGraph lectures and presentations must be loaded from, saved and exported into persistent forms, such as AudioGraph, Scrapbook and web-ready file formats. Figure 5.16 illustrates the file building class diagram.

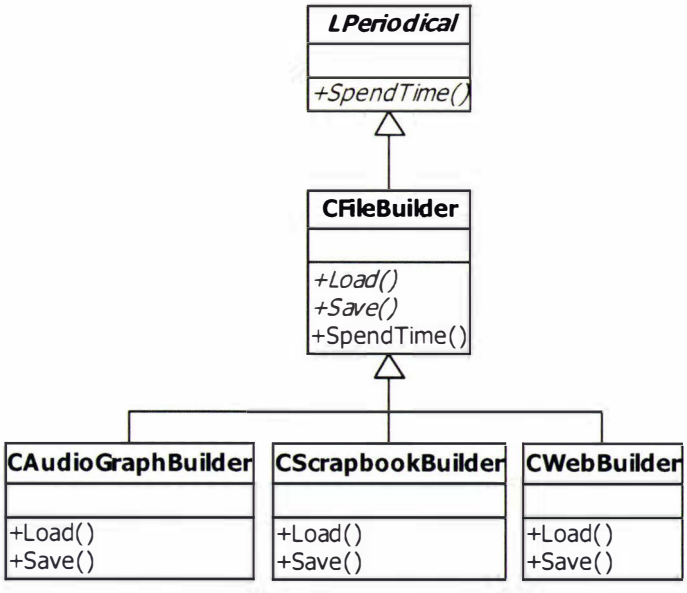


Figure 5.16. File building class diagram

Whenever a *structural hierarchy* — as defined in Section 4.3.2 — needs to be externalised, a number of alternatives are available:

- The instances contained in the hierarchy may be given the responsibility to stream themselves into and out of a persistent form. In such a situation, the structure of the output stream typically matches the order in which the structural hierarchy is streamed, and contains only state data for the structural hierarchy itself. This approach has the benefit that streaming knowledge is encapsulated within the persisted classes, but lacks flexibility of structure for the input or output streams, unless separate special supporting mechanisms are developed.
- The responsibility for streaming may be assigned to a class or set of classes that may dictate a more complex structure of the output stream, potentially including elements that lie outside the structural hierarchy. This approach has the benefit that stream structure is controlled in a centralised fashion, but may require intimate knowledge of the persisted classes in order to stream all the necessary state data.

For the AudioGraph recorder, the solution we have chosen is closer to the second option outlined above, due to the different demands of the AudioGraph (**`CAudioGraphBuilder`**), Scrapbook (**`CScrapbookBuilder`**) and AudioGraph web-ready (**`CWebBuilder`**) file formats. We have adopted a variation of the *builder* design pattern [127] (pp. 97) orchestrated through a realization of the *template method* design pattern [127] (pp. 325). The approach is discussed in more detail in Sections 5.7.4 and 5.7.6.

We made this choice since it offered us the possibility to use a common framework (**`CFileBuilder`**) for three substantially different file formats: the AudioGraph native lecture document format, the Scrapbook file format and the AudioGraph web-ready file format.

5.4.3 Deployment View

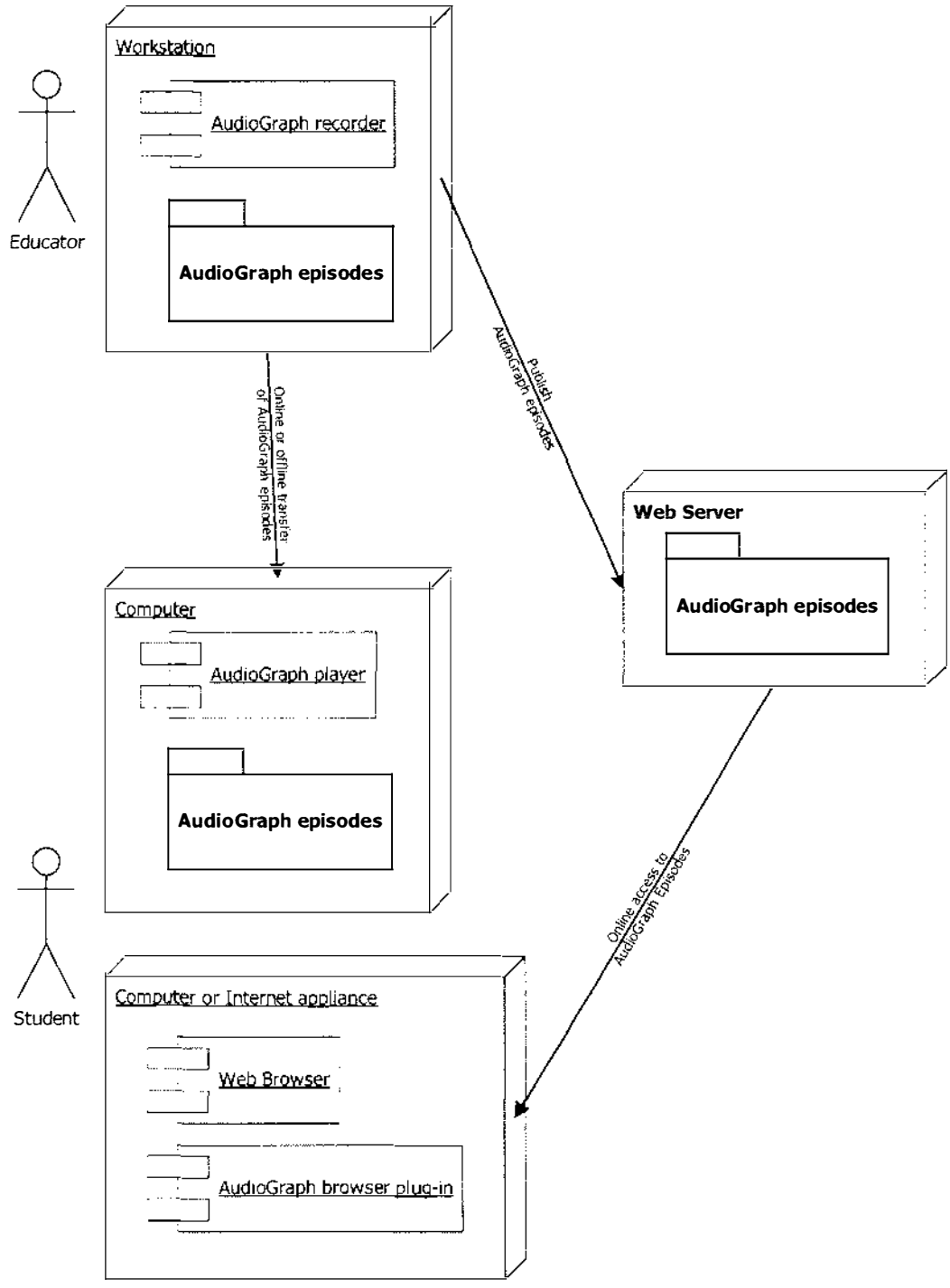


Figure 5.17 AudioGraph deployment diagram

Figure 5.17 represents the typical deployment of applications in the AudioGraph suite.

5.4.4 Performance

One significant advantage to using a hierarchical approach for rendering visual component hierarchies is that no special memory buffer is required to store the state of the visual hierarchy.

As areas of the host view need to be refreshed, the code calls upon the affected annotations to redraw themselves in the appropriate regions. Although this uses less memory, it is not as fast to render the appropriate view as ultimately possible.

For typical views, where the number of panes is relatively small and their visual status is relatively static, this approach is best suited, as it places no significant demands on memory and its performance is perfectly satisfactory.

On the other hand, the approach of using memory buffers to store the visual representation of views is not at all efficient in terms of memory usage, should all views be forced to use it, and as a result cannot be adopted universally. We see here another example of the trade-off between use of memory and performance.

In the solution adopted in the AudioGraph recorder, at the very beginning of a presentation, all panes of the annotations in the current episode are hidden, which means they have no visual effect. As the playback of an AudioGraph presentation unfolds, the panes of the corresponding annotations must be made visible. The typical approach to make a pane visible in the PowerPlant [148] framework is to change its visibility status and refresh the views that contain it in order to display it properly, according to its z-order.

Although this approach is correct, it has the drawback of requiring a traversal of all annotations in order to accomplish it. As playback progresses, for each annotation that is displayed, all annotations are traversed and the visible ones are redrawn, leading to $O(N^2)$ complexity in playback if no optimisations are performed.

It is clearly desirable to have a consistent playback experience rather than having presentation playback slow down as an ever-increasing number of annotations are redrawn with each new annotation being displayed.

As a result, a number of optimisation approaches must be used in order to improve the performance of the rendering mechanism. For the AudioGraph recorder, the first one involves adapting the `Show()` method to no longer require the refresh of the enclosing views, as the normal PowerPlant [148] method does, and merely change the visibility status and render the required visual representation.

This approach is possible in our context, since at all times the latest graphical annotation being displayed is at the top of the z-order. However, it requires that pictures be dealt with in a special manner, since in the AudioGraph visual model they are always displayed in the background, underneath all graphical annotations, and therefore appear lower in the z-order compared to all other graphical annotations.

As a result, it would be desirable to be able to display all pictures as a unit, as the first element in the z-order. This is accomplished by using an off-screen view that captures the image data for all pictures, and can be copied to screen very efficiently. This background off-screen view is initialised with the background colour of the episode, and it is displayed before all other annotations. There are still drawbacks to this approach: as picture annotations are displayed, since they must request that the first element in the z-order be redrawn, all annotations are traversed and many of them may have to be redrawn, leading to performance degradation. In addition, in the presence of highlight annotations, since the effect of the visual transformation required to obtain the highlight — a bitwise *and* operation — is not reversible, it is necessary to redraw the other annotations as above.

For the `Hide()` method, it is not easily possible to escape without having to refresh the area covered by the annotation being hidden, and therefore the performance of this operation is relatively poor. Thankfully, this method is not required during playback, and therefore its impact is not so harmful.

These performance deficiencies have plagued the initial versions of the AudioGraph recorder and players. In order to address these issues and obtain `O(1)` playback performance, special measures have to be taken, as will be clarified in Section 5.5.2.2, which outlines the approach taken in version 2.0 of the AudioGraph browser plug-in.

5.5 Browser Plug-in Architecture

The browser plug-in is the software component used to view episodes on-line, once they have been exported in the AudioGraph web-ready file format.

5.5.1 Use-Case View

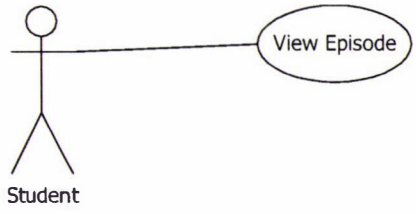


Figure 5.18 Browser plug-in and Player use-case diagram

Figure 5.18 shows the browser plug-in and player use-case diagram, which is very simple.

Students use the browser plug-in or the player application to view web-ready episodes. In version 2.0, the use-case also covers hyperlinked episodes presented in the same player instance.

5.5.2 Logical View

In the following sections, we will contrast the class diagrams of the browser plug-ins written for versions 1.0 and 2.0 of the AudioGraph file format specification.

5.5.2.1 Version 1.0

In the initial version of the browser plug-in for the Macintosh, the rendering engine used was virtually identical to the one used in the AudioGraph recorder. For reasons outlined above in Section 3.6, this approach was less than ideal from a performance perspective in the playback context, as in certain cases polynomial performance degradation would be manifest. Figure 5.19 presents the class diagram for version 1.0 of the browser plug-in.

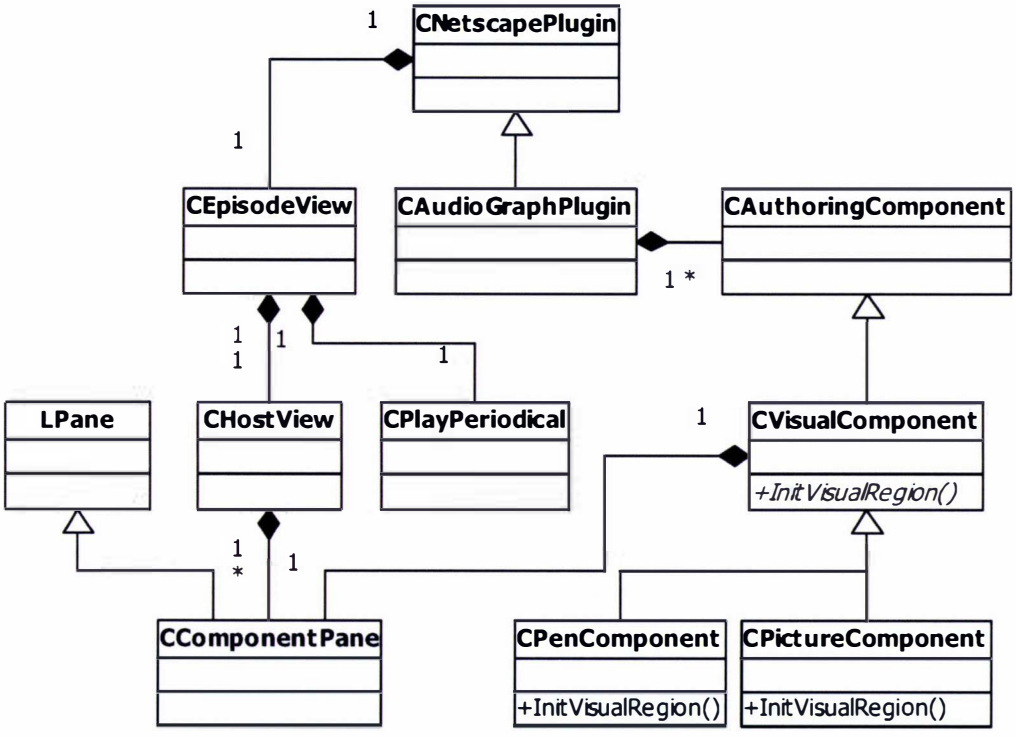


Figure 5.19. Class diagram for version 1.0 of the browser plug-in

This version also did not implement clear-screen, linking or halt annotations, as they were not yet specified at the time. As a result, the architecture of the plug-in is focused on playback of a single episode rather than multiple episodes, as would be required in the presence of links.

You may note the same structural hierarchy as in the AudioGraph recorder, with the `CHostView` containing `CComponentPane` instances associated with visual components (`CVisualComponent`). Other familiar classes are present as well, such as the `CPenComponent`, `CPictureComponent` and `CAuthoringComponent`.

However, in contrast with the approach taken in the recorder application, where the various graphical annotations have their own types, the web-file format simulates the effect of all the different annotations by judiciously using the pen annotation (**CPenComponent**) and associated attributes.

As a result, the pen annotation is the only type of annotation required to provide all graphical foreground effects. For instance, to simulate a line annotation, a pen annotation (**CPenComponent**) with only two points can be used. To simulate a rectangle, a pen annotation with four points and the *closed* attribute may be used. The *filled* attribute may be used as well to indicate a filled rectangle. For arc and oval annotations, four points are used to indicate the bounding rectangle and the start and end-points of the desired arc, and a special *ellipsoidal* drawing attribute is set. The pen annotations represent themselves directly, as specified in the recorder.

Notice how the plug-in (**CAudioGraphPlugin**) plays the role of the episode in this design. It extends the **CNetscapePlugin** class, which provides the *adapter* [127] (pp. 139) between the browser and the custom code that implements the plug-in functionality. It also hosts the view (**CEpisodeView**) where all display activities occur. The **CHostView** and **CComponentPane** classes are virtually identical to their recorder counterparts.

Similarly, the **CPlayPeriodical** instance plays the same role it does in the recorder, controlling the progress of the playback, except that in the absence of the edit console, it must only update the progress bar, as seen in Figure 3.17.

5.5.2.2 Version 2.0

The second version of the browser plug-in has a customised visual architecture, designed primarily for speed. In addition, it supports multiple episodes in the same plug-in instance. Figure 5.20 represents its class diagram.

Note the absence of the **CComponentPane** instances. In this version of the plug-in drawing occurs directly in the **CHostView**, using an efficient rendering approach discussed in more detail in Section 5.5.4. The **Behaviour** instances dictate the **CPenComponent** object display characteristics, such as **Normal**, **Erasing** or **Highlighting**, in an implementation of the *strategy* design pattern [127] (pp. 315)

Since this version of the plug-in must support multiple episodes, the earlier approach of having the plug-in class host all annotations came under review. As the specification of the linking annotations indicates that episodes are to be dealt with atomically, it proved more advantageous

to structure the new architecture around episodes (`CEpisode`) and an episode manager (`CEpisodeMgr`) that would provide seamless playback and linking services.

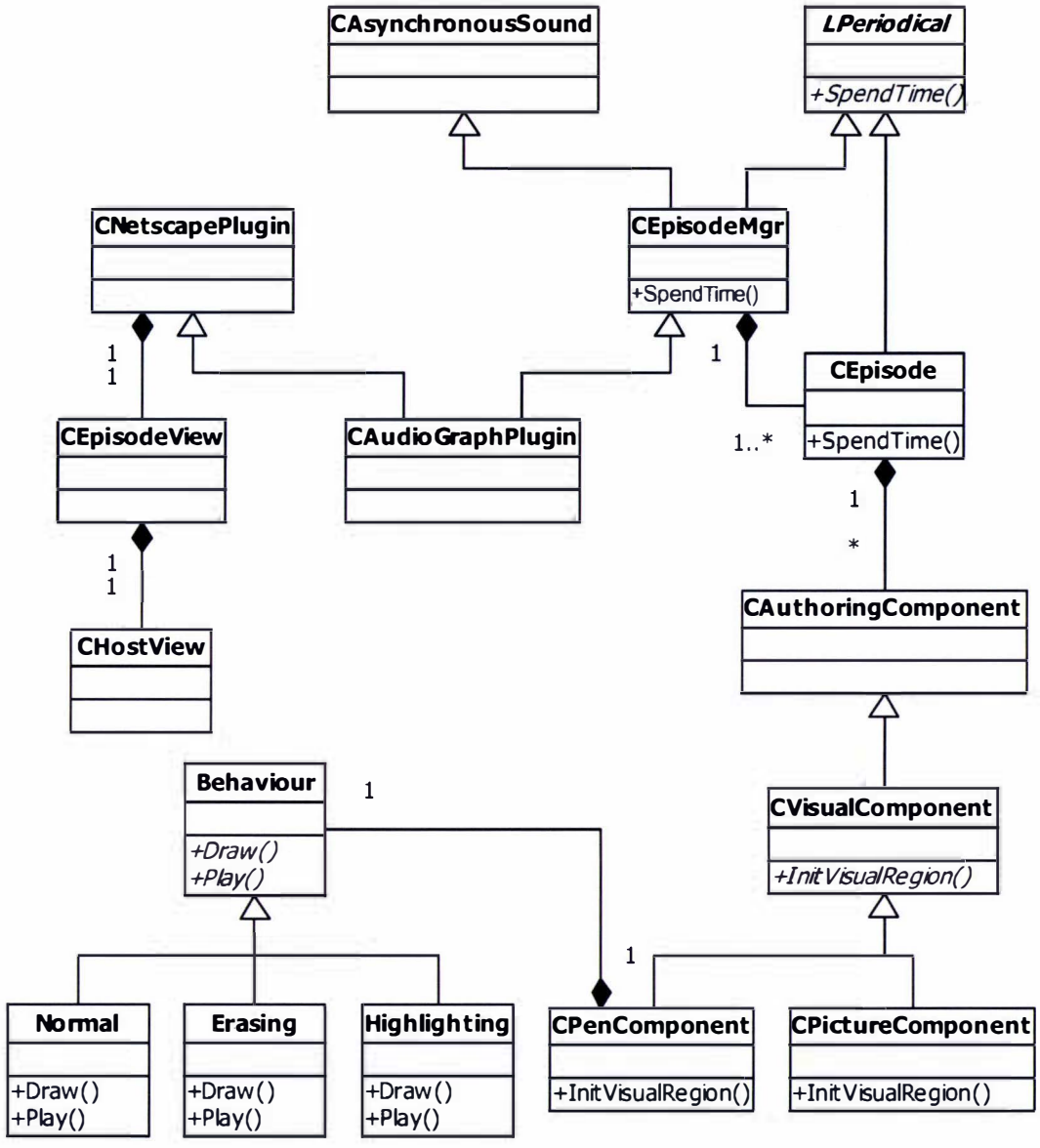


Figure 5.20. Class diagram for version 2.0 of the browser plug-in

By atomic episode management in the presence of links, we mean that once a link to another in-instance episode is taken, all the subsequent *episodes* are discarded rather than all *annotations* after the link itself. In other words, if there are any annotations following the link annotation in the current episode, they will be preserved and the annotations of the freshly linked episode will be made available in the playback stream after the end of the current episode.

In such a context, an option might have been to perpetuate the original approach, keeping a single annotation container, complemented with suitable data structures to indicate episode boundaries. However, episodes are substantially more complex than a mere collection of

annotations, and therefore episode instantiation and cleanup would have meant more than just adding annotations into the common container and marking boundaries.

Such issues can be dealt with in an elegant fashion by designing a suitable episode class (**CEpisode**). However, for reasons of data encapsulation, once an episode class is created, its associated annotations would be best hosted by the episode itself rather than in an external, shared container. However, from a presentation perspective, the sequence of annotations must be seamless when switching between episodes, even in the presence of multiple containers.

To provide this capability, an episode manager class (**CEpisodeMgr**) can orchestrate the sequence of episodes and provide a clear *iterator* interface on an aggregate level, which hides the storage complexities involved in using multiple annotation containers. The relevant *iterator* design pattern [127] (pp. 257) is discussed in more detail in Section 4.3.4.

As can be seen in Figure 5.20, we have chosen to make the AudioGraph plug-in class (**CAudioGraphPlugin**) an episode manager (**CEpisodeMgr**), which may control one or more self-contained episodes (**CEpisode**). To provide seamless access to annotations, the episode manager maintains an *iterator* on the episode container. Each episode, in turn, maintains an *iterator* on the annotation collection that it hosts. As a result, when iterating through the annotations in the overall presentation, most of the iterations happen on the episode level, and occasionally the episode manager *iterator* advances as episode boundaries are encountered. This approach combines efficiency with good encapsulation characteristics, which increase application robustness.

The *periodical* functionality (due to inheritance from **LPeriodical**) in the episode (**CEpisode**) instances is used to determine whether the source media file checksums are correct. If they are not, presentation is summarily aborted, as it indicates either file corruption or tampering.

The *periodical* functionality in the episode manager (**CEpisodeMgr**) instances is used to provide playback capabilities, unlike the approach taken in the version 1.0 of the plug-in, which used a **CPlayPeriodical** instance associated with the episode view, as can be seen in Figure 5.19.

The mechanism used for playback in version 1.0 of the plug-in was very similar to the one used in the recorder application. However, it seemed more appropriate to associate playback responsibilities with the episode manager class rather than the view class, for a tighter integration of responsibilities. Similarly, in the recorder application, playback capabilities could be associated with the episode or even lecture instances rather than the associated views.

In addition, for each episode manager version 2.0 uses dedicated asynchronous sound services (`CAsynchronousSound`), allowing multiple presentations to unfold and play sounds simultaneously rather than using an asynchronous sound *singleton*, which enabled a single presentation to play sound annotations at any one time.

5.5.3 Deployment View

Figure 5.17 also captures the typical deployment structure for the player applications.

5.5.4 Performance

You may have noticed in Figure 5.20 that no panes (such as the `CComponentPane`) are associated with the visual components (`CVisualComponent`) anymore. In contrast with the version 1.0 of the plug-in, the rendering engine in version 2.0 more closely resembles the AudioGraph visual model, as shown in Figure 3.1. The *host view* contains *background*, *screen* and *alpha-channel* off-screen buffers, as opposed to merely a *background* off-screen buffer in version 1.0.

Annotations affect these three off-screen buffers as discussed in Section 3.2 and described in the web-file format specification. During playback, the *host view* then copies, as required, regions of the *screen* off-screen buffer on the screen and draws any required highlight annotations. Although this approach uses more memory for the extra off-screen buffers, it provides better playback performance, since it achieves amortised `O(1)` rendering complexity.

The pen annotation rendering mechanism used in version 2.0 of the plug-in is interesting in that it uses a variant of the *strategy* design pattern [127] (pp. 315) in the successful pursuit of increased efficiency.

Previously, whenever a pen annotation was rendered, decisions were made at run-time on the appropriate effect the annotation should trigger, based on the values of the *drawing mode* attribute such as *normal*, *erasing* or *highlighting*. This meant that if an annotation must be rendered multiple times, the same decisions would be made repeatedly. As decisions are relatively expensive in modern microprocessor architectures, this situation only aggravated the performance profile of the solution.

Fortunately, an elegant solution to this issue exists. The specific rendering and playback behaviours for the various drawing modes can be encapsulated in specific classes, which can be associated with the pen annotation instances at creation time. From then on, at run-time, no further decisions are required, since each annotation instance would use the appropriate behaviour mechanism, as dictated by its attributes. The fact that annotation attributes do not

change in the plug-in application means that the initial behaviour choices may be kept unchanged throughout the lifetime of the annotations.

All player applications of version 1.0 on the Macintosh platform have used the same visual hierarchy as the 1.0 recorder application. However, what has been most valuable for an editing application, where users had to be able to select and move graphical annotations around in the editing stream, which required their visual relationships to be adjusted accordingly, has proven to be an unsuitable choice for playback applications, where the playback stream is clearly defined and immutable.

The playback applications did not require the unhindered flexibility of the recorder application. However, high performance was a very desirable trait for them. Alas, due to the use of the same visual hierarchy as in the recorder, most drawing operations during playback were incurring linear — $O(N)$ — and in certain cases even polynomial — $O(N^2)$ — performance degradation leading to unacceptable performance for large presentations.

In the original version of the AudioGraph rendering engine, borrowed from the recorder application, the visual hierarchy was based on the *composite* design pattern [127] (pp. 163), as can be seen in Figure 5.19 on page 172. During playback, each time a new annotation was displayed, the code had to traverse the collection of `CComponentPane` instances associated with the graphical annotations, in order to draw it in the appropriate fashion. This situation leads to $O(N)$ complexity when rendering normal graphical components.

The situation was further deteriorated when highlighting components and vanishing images were taken into account. In such cases, not only was the code required to traverse the collection of panes in order to render the annotations themselves, but also since a number of other sibling annotations typically had to be redrawn as well, the complexity was raised by another order of magnitude, leading to $O(N^2)$ performance degradation.

With a view to overcome these limitations, the overriding goal for the 2.0 version of the player applications has been to ensure that as many playback operations as possible are executed in amortised constant time — $O(1)$.

We have succeeded in developing AudioGraph web browser plug-ins running under Apple Macintosh OS 9 and under OS X [160] — using the Carbon [161] interface — that implement the 2.0 AudioGraph web-ready file format specification almost fully and satisfy the above performance requirement. The only unsupported feature of the specification is support for the Ogg Vorbis file format [242] since at the time of writing the required software libraries were not yet advanced enough.

5.6 Player Architecture

For version 1.0 of the web file format specification, the author has also developed a standalone player application, which provides web-ready file playback capability without requiring a web browser and associated web pages.

5.6.1 Logical View

The architecture of the standalone Macintosh player application is very similar to version 1.0 of the browser plug-in architecture described in Section 5.5.2.1. Figure 5.21 illustrates the player's class diagram.

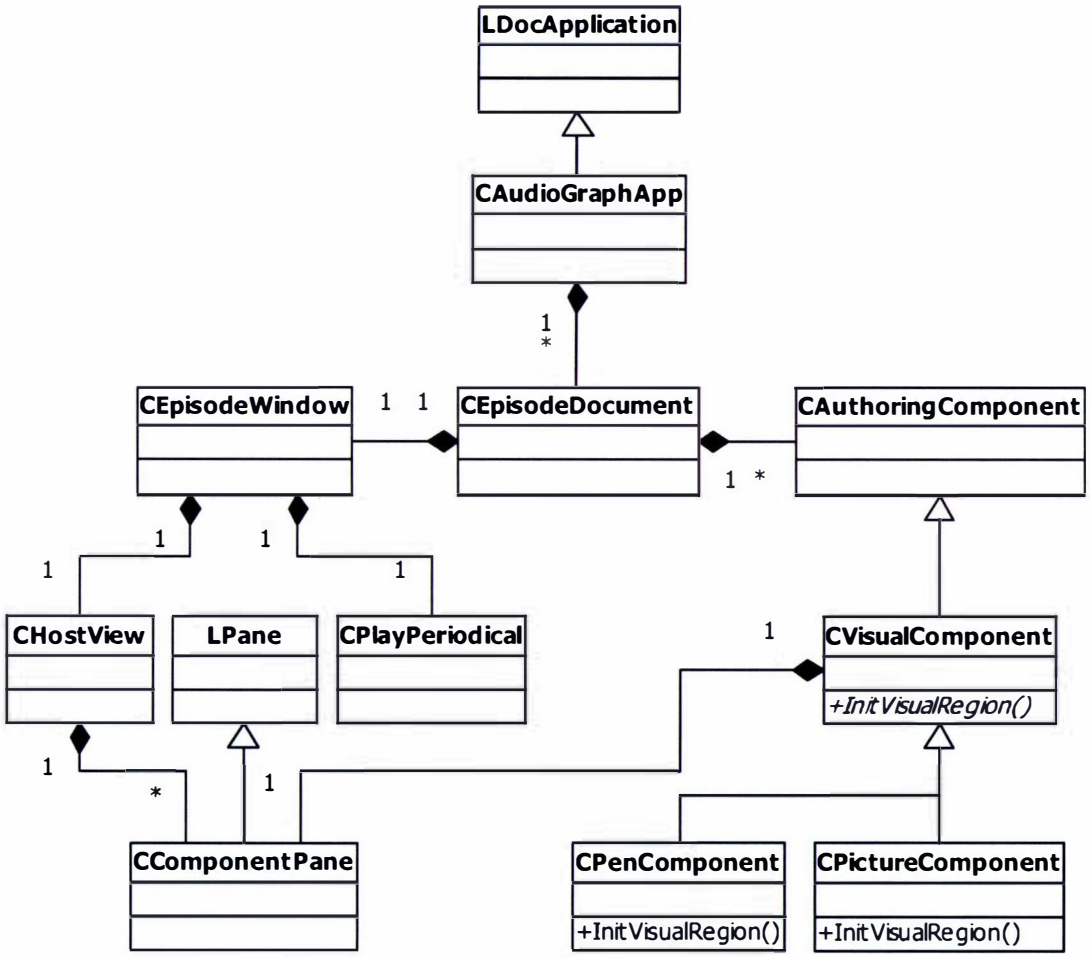


Figure 5.21. Class diagram for the stand-alone Macintosh AudioGraph player

The main difference lies in the fact that the standalone application (`CAudioGraphApp`) must provide its own application framework (hence the use of `LDocApplication`), since it cannot rely on the browser plug-in framework.

As version 1.0 did not attempt to support multiple linked episodes, the episode document (**CEpisodeDocument**) focuses on a single episode, and provides virtually identical functionality to the plug-in class (**CAudioGraphPlugin** in Figure 5.19).

The episode window (**CEpisodeWindow**) replaces the episode view (**CEpisodeView** in Figure 5.19), as the player must use a native Macintosh window. Beyond that aspect, the episode window and the episode view in the plug-in provide identical functionality to the application.

As the rendering engine in the player is virtually identical with the version 1.0 plug-in and recorder application, it suffers from the same playback performance limitations, as outlined in Section 5.4.2.3.3.

To date, demand for a standalone Macintosh player has not been so significant as to warrant developing a new version that would take advantage of the outstanding performance advances achieved in version 2.0 of the plug-in. As tighter future integration of AudioGraphic content into portals or Learning Management Systems is likely to increase, a stand-alone player might not be required ever again.

5.6.2 Deployment View

Figure 5.17 also captures the typical deployment structure for the player applications.

5.7 Design Patterns Applied

In the following sections, we will examine some of the design patterns that have had a significant impact on the architecture of the AudioGraph suite of applications.

5.7.1 Composite — Usage Examples

In the AudioGraph recorder, we use the *composite* design pattern [127] (pp. 163) for managing authoring components. Figure 5.22 shows the class diagram of the authoring component class hierarchy in version 1.0.

Notice that the **CGroupComponent** class allows us to store an arbitrary number of components within a component, without any restrictions apart from system resources. All other classes in the hierarchy are leaf classes with respect to the *composite* design pattern [127] (pp. 163). Besides leaf class instances, **CGroupComponent** instances can also have other **CGroupComponent** instances as children, giving users the freedom to aggregate components in their presentations as they best see fit.

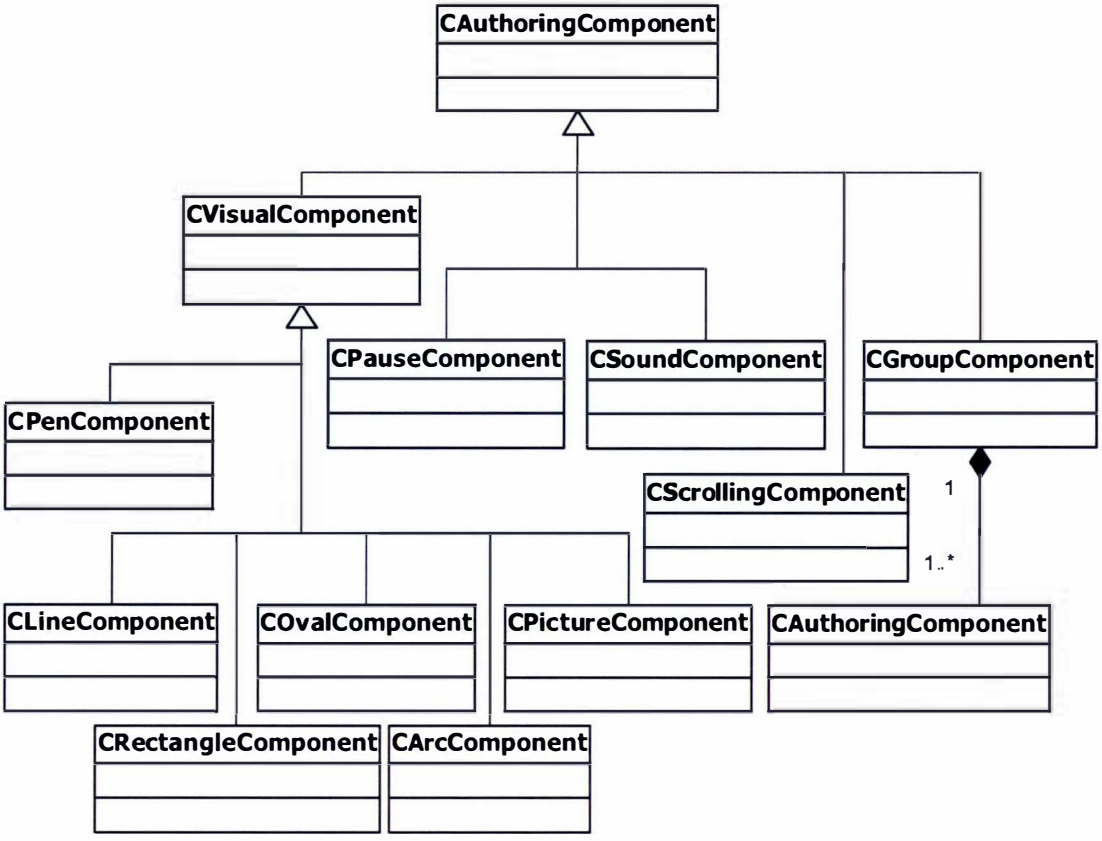


Figure 5.22. AudioGraph recorder component class hierarchy

The PowerPlant framework [148] makes use of the same design pattern for the visual hierarchy classes such as **`LPane`**, **`LView`**, and **`LWindow`**. The motivation is to have **`LPane`** focus on frame information and **`LView`** on visual structure. Therefore, **`LView`**'s are allowed to have children panes (which may be views themselves) and **`LPane`**'s are leaf classes. Figure 5.23 illustrates the class diagram of the core PowerPlant [148] visual elements.

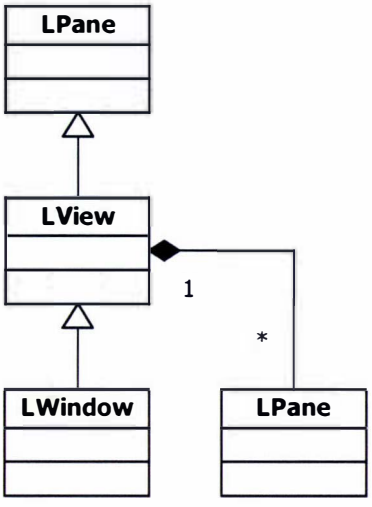


Figure 5.23. PowerPlant core visual elements class hierarchy

Should all visual elements have had the capability to host children, in the vast majority of cases a lot of memory would have been wasted by leaf views that would never have held children. By splitting the responsibilities between the **`LPane`** component and the **`LView`** container, better efficiency in modelling visual hierarchies is achieved.

5.7.2 Iterator — Usage Examples

Given that the use of iterators is so ubiquitous, most usage examples would be relatively trivial. However, as described in Section 5.5.2.2, in the 2.0 version of the AudioGraph plug-in, we have developed a uniform iteration mechanism across multiple annotation component containers hosted in separate episode instances.

In another example well worth mentioning, we have put the *iterator* design pattern [127] (pp. 257) to good use in significantly improving the performance of manipulating visual regions in version 2.0 of the AudioGraph browser plug-in for Macintosh.

The core of the issue lies in the layering architecture required to implement transparency. For drawing arbitrarily complex pen components, simple Macintosh Toolbox routines that deal with regions are convenient and efficient.

However, when it comes to recording 8-bit alpha values for all pixels in the visible screen area, simple Macintosh Toolbox routines are no longer sufficient. Normal drawing operations in 8-bit graphic ports would involve multiple transformations of image indexes into and out of RGB colour values, and would be too unwieldy and inefficient to manage in order to achieve the calculations we require. Should we perform all calculations in a 32-bit graphic port, we would waste a lot of memory and processing power and not achieve the desired efficiency.

As a result, we must use a simple array of bytes to record the alpha values and need a fast means of updating individual pixel values in this buffer based on the bit values of the black-and-white bitmaps used by the visual components to describe their visual regions.

Traversing the bounding rectangles of such monochrome bitmaps and interrogating every single pixel to determine whether it is zero or one would be inefficient in most cases.

When the visual region is the full bounding rectangle, we need not interrogate any pixels at all. We can simply iterate through all the pixels in the region and perform the required calculations.

When the visual region is any form of diagonal line, or even a more intricate pen component composed of sequences of lines, iterating through all pixels in the bounding rectangle is also likely to be inefficient as most of them will probably be zero.

At first glance, it might seem that there is no alternative to improve this approach. Fortunately, testing bytes for a zero value is fast, and it replaces eight individual bit tests at a time.

Developing this idea, we can create a region iterator that efficiently scans any region. We initialise the region iterator by supplying it with the native Macintosh region handle, and wish to find out, one at a time, the (x, y) coordinates of all pixels in the region.

A typical usage example would be:

```
Rect        frameRect;
CalcLocalFrameRect(frameRect);

StRegionIterator   iterator(inRgnH, &frameRect);
SInt32      x, y;
while(iterator.Next(x, y)) {
    // perform processing
}
```

If the region overlaps the frame rectangle entirely, then using the iterator is going to be very close to being as fast as using the following simple set of loops:

```
for( SInt32 x = frameRect.left; x < frameRect.right; ++x) {
    for( SInt32 y = frameRect.top; y < frameRect.bottom; ++y) {
        // perform processing
    }
}
```

The **`StRegionIterator::Next()`** method section dealing with such rectangular regions follows:

```
bool StRegionIterator::Next(SInt32 &outX, SInt32 &outY) {
// If we're dealing with a square region (fastest)
//
    if(mIsSquare) {
    // We are on top of the next bit to pass out if we didn't
    // pass beyond the bounds on the previous call
    //
        if(mX < mBounds.right) {
            outX = mX;
            outY = mY;
            ++mX;       // Go to the next X
            return true;
        }

    // Line exhausted, must go to the next line, next Y
    //
        mX = mBounds.left;
        ++mY;

    // If not past the bounds, we can pass the next point
    //
        if(mY < mBounds.bottom) {

            outX = mX;
            outY = mY;
            ++mX;       // Go to the next X
            return true;
        }
        outX = outY = 0;
        return false;
    }
    ...
}
```

Notice that apart from the test for **`mIsSquare`**, the above code performs the same actions as the simple set of loops, as they share the same instructions apart from the assignments to the output parameters.

Modern compilers may even optimise away the cost of assignments such as these:

```
outX = mX;
outY = mY;
```

By using register variables, leading to very good performance.

If the region does not overlap the frame rectangle entirely, then a monochrome bitmap corresponding to the intersection of the region and the frame rectangle is built, and the iterator starts from the top left corner and proceeds horizontally to search for the next available pixel with a value of one, skipping eight bits at a time when bytes have a zero value, as follows:

```
// If we must check the bitmap, we look to see whether we've skipped
// past the row bounds
//
    if(mX < mBounds.right) {
    // Find the next bit set on this line
    //
        if(Found()) {
            outX = mX;
            outY = mY;
            NextX();
            return true;
        }
        // Exhausted this line, we must skip to the next
    }

// Line exhausted, must go to the next line, next Y
//
    NextY();
    while(mY < mBounds.bottom) {
    // Search on this line
    //
        if(Found()) {
            outX = mX;
            outY = mY;
            NextX();
            return true;
        }

    // Exhausted this line as well, go to the next
    //
        NextY();
    }
    outX = outY = 0;
    return false;
```

Notice that the processing steps are very similar to the rectangular special case, apart from the bit existence tests and specialised forms of increment. The bit existence test:

```
// Search for the next bit set on the current row, from the current position onwards.
// Update the position on the row as we go along
//
    bool Found() {
        do {
            if(mByte & mBit) return true;
            NextX();
        } while (mX < mBounds.right);
        return false;
    }
```

Will call upon the special increment method, **`NextX()`**, to perform the fast increments along the current raster line.

Here follows its implementation:

```
void NextX() {  // Advance to the next bit in the current row
// Skip zero bytes quickly
//
    if(0 == mByte) {
    // Keep mX and mBit in sync
    //
        if(mBit != kBit0) {
            ++mBytePtr;
            mByte = *mBytePtr;
            while(mBit != 0) {
                ++mX;
                mBit >>= 1;
            }
            mBit = kBit0;
            if(!(0 == mByte && mX < mBounds.right)) return;
        }

    // Skip one byte at a time
    //
        do {
            ++mBytePtr;
            mByte = *mBytePtr;
            mX += 8;
        } while (0 == mByte && mX < mBounds.right);
        return;
    }
// Advance to the next x coordinate
//
    ++mX;
    mBit >>= 1;
    if(0 == mBit) {
        mBit = kBit0;
        ++mBytePtr;
        mByte = *mBytePtr;
    }
}
```

Note that the above method does not promise to advance precisely to the next bit with a value one. It simply advances to the next available bit (if the current byte is non-zero). Otherwise, it first synchronises the bit mask **`mBit`** and the coordinate counter **`mX`** (in order to be able to proceed interrogating correctly the next byte in the bitmap) and then skips in hops of eight bits until a non-zero byte is found again or the right boundary is reached. The interplay with the method **`Found()`** ensures that a sequence of bit tests in the non-zero byte follows the fast skipping to find the sought after next bit that is set.

The **`NextY()`** method resets the iterator's state variables to the appropriate values required for iterating over the next raster line:

```
void NextY() {  // Advance to the next row
    mX = mBounds.left;
    ++mY;
    mRowPtr += mRowBytes;
    mBytePtr = mRowPtr + mFirstPixelByteOffset;
    mByte = *mBytePtr;
    mBit = mFirstPixelBit;
}
```

Tests have shown this approach to give significantly better performance than testing all the individual bits. The availability of this mechanism has improved the overall performance of the browser plug-in rendering engine considerably, reducing the cost of providing full eight-bit alpha channel support.

5.7.3 Memento — Usage Examples

Most modern application development frameworks provide a mechanism to build windows and all their associated controls from definitions stored in resource files, which, in effect, is a manifestation of the *memento* design pattern [127] (pp. 283). The PowerPlant framework [148] uses this design pattern intensively, with support from the Metrowerks modelling tool *Constructor* [147]. For instance, *Constructor* offers a WYSIWYG interface for creating visual interface elements, and stores the associated descriptions in **`PPob`** resources. Applications built with PowerPlant [148] support can then make use of these **`PPob`** resources to easily recreate the desired window, dialog box or view, no matter how complex the definition.

In a certain sense, the AudioGraph suite uses this design pattern too, but in a more controlled fashion, as the file formats used have structures that are richer than mere persistent object containers. The AudioGraph recorder externalises the state of lecture objects, episode objects and components by saving them in a native format. The lecture files are then used to re-create the structural hierarchy once they are opened for editing in the recorder.

The AudioGraph file format is also used to externalise individual AudioGraph episodes in a web-ready format. The AudioGraph player and browser plug-ins use this representation to reanimate the encoded structural hierarchy.

The *memento* design pattern [127] (pp. 283) is also used indirectly in implementing the multiple undo functionality, by integrating it with the *command* design pattern [127] (pp. 233).

5.7.4 Template Method — Usage Examples

In the AudioGraph recorder, it quickly became apparent that the application needed to cope with a number of different file formats, for both loading and storing information. However, the main processing steps required would have been very similar in all instances. Therefore, the *template method* design pattern [127] (pp. 325) was a clear candidate for a robust and flexible implementation that would orchestrate the steps required in streaming data to and from the supported formats.

As can be seen in Figure 5.16, the AudioGraph, *scrapbook* and *web builder* classes derive from the **`CFileBuilder`** class.

This is also a periodical, used as a *repeater*, and its **SpendTime()** method orchestrates the streaming process:

```
void CFileBuilder::SpendTime(const EventRecord &inMacEvent) {
    try {
        switch( mAction ) {
            case fba_Loading:
                ProcessLoad();
                break;
            case fba_Saving:
                ProcessSave();
                break;
            case fba_Dormant:
                SetState( fbs_Done );
                ...
                StopRepeating();
                break;
        }
    } catch ( inErr ) {
        mAction = fba_Dormant;  // Go into Dormant behaviour
        switch(inErr) {

            case err_IncorrectTag: {
                ...
                break;
            }
        ...
        }
    }
}
```

The **ProcessLoad()** and **ProcessSave()** methods act as state machines, invoking *template methods* like **LoadLecture()**, **LoadEpisode()** and **LoadComponent()**:

```
void CFileBuilder::ProcessLoad() {
    switch( GetState() ) {
        case fbs_LoadingLecture:
            LoadLecture();
            SetState( fbs_LoadingEnds );
            break;
        case fbs_LoadingEpisode:
            LoadEpisode();
            if( FinishedLoadingEpisode() ) {
                SetState( fbs_LoadingEnds );
            } else {
                SetState( fbs_LoadingComponent );
            }
            break;
        case fbs_LoadingComponent:
            LoadComponent();
            if( FinishedLoadingComponents() ) {
                SetState( fbs_LoadingEnds );
            }
            break;
        case fbs_LoadingEnds:
            if( FinishedLoadingLecture() ) {
                SetState( fbs_Done );
                mAction = fba_Dormant;  // Indicate that processing must end
                GetDocument().SetIsBusy( false );
                DoneLoading();
            } else {
                SetState( fbs_LoadingEpisode );
            }
            break;
        ...
    }
}
```

In this context, the individual builder classes implement the *template methods* to provide the adequate behaviour for the particular type of stream that they are designed to operate on. The

abstract file builder base-class merely provides the framework in which the concrete builders will operate.

In retrospect, the file builder periodical processing could be re-factored in the spirit of the *strategy* design pattern [127] (pp. 315), adopting an approach similar to the one used for the pen annotation rendering behaviours as implemented in version 2.0 of the plug-in, and thus eliminating the need for the `switch` statement and improving performance.

Since the builder action is known for the duration of the load or save operation, instead of making the decision to execute `ProcessLoad()` or `ProcessSave()` each time control is given to `SpendTime()`, we could invoke the appropriate function through a pointer to the member function. This pointer can be set at the beginning of the operation, and it would only be changed when the operation ends, as the same method is repeatedly called until all the components are processed.

5.7.5 Strategy — Usage Examples

We have already explained in Section 5.5.4 how the pen-annotation rendering behaviours are implemented as interchangeable strategies.

In addition, on a higher level of abstraction, the different file builders are also used as streaming strategies. When a streaming operation is invoked, the appropriate file builder instance is created, and it then carries out the operation over time, with the aid of the periodical invocations provided by the base file builder.

5.7.5.1 An Interesting Variation

In the interest of increasing the performance of rendering PNG [218] images — with full alpha-channel support — the author designed an interesting variation of the *strategy* design pattern [127] (pp. 315) in the pixel-by-pixel rendering mechanism used in version 2.0 of the plug-in.

The problem consisted in the fact that the screen buffer may have various pixel depths, as dictated by the system pixel depth. As a result, the rendering algorithm was faced with different pixel representations, although the processing required to compute the correct pixel values was the same in all cases.

To resolve the issue, we chose to provide a number of pixel-representation adapter classes, with appropriate arithmetic operators for the various pixel formats, such as 32-bit RGBA and 16-bit RGB. For instance:

```
//=============================================================================
//  MacRGBQuad32
//=============================================================================
// Perform operations on a 32-bit quad in one go
//
    class MacRGBQuad32 {
        UInt8       unused;
    public:
        UInt8       r;
        UInt8       g;
        UInt8       b;
        MacRGBQuad32(const MacRGBQuad32 &inQuad) :
            r(inQuad.r), g(inQuad.g), b(inQuad.b)
        {}
        MacRGBQuad32(const RGBAQuad &inQuad) :
            r(inQuad.r), g(inQuad.g), b(inQuad.b)
        {}
        MacRGBQuad32(const UInt8 inR, const UInt8 inG, const UInt8 inB) :
            r(inR), g(inG), b(inB)
        {}
        MacRGBQuad32 &operator =(const MacRGBQuad32 &inQuad) {
            r = inQuad.r;
            g = inQuad.g;
            b = inQuad.b;
            return *this;
        }
        MacRGBQuad32 &operator =(const RGBAQuad &inQuad) {
            r = inQuad.r;
            g = inQuad.g;
            b = inQuad.b;
            return *this;
        }
        MacRGBQuad32 &operator =(const RGBColor &inRGB) {
            r = inRGB.red;
            g = inRGB.green;
            b = inRGB.blue;
            return *this;
        }
        MacRGBQuad32 operator +(const MacRGBQuad32 &inQuad) {
            return MacRGBQuad32(r + inQuad.r, g + inQuad.g, b + inQuad.b);
        }
        MacRGBQuad32 operator -(const MacRGBQuad32 &inQuad) {
            return MacRGBQuad32(r - inQuad.r, g - inQuad.g, b - inQuad.b);
        }
        MacRGBQuad32 operator *(const double inValue) {
            return MacRGBQuad32(r * inValue, g * inValue, b * inValue);
        }
        MacRGBQuad32 operator /(const double inValue) {
            return MacRGBQuad32(r / inValue, g / inValue, b / inValue);
        }
    };
```

Using these supporting classes, a template member-function could provide the rendering algorithm, very clearly expressed in terms of the arithmetic operators, and the main rendering method would invoke the appropriate strategy by invoking the template member-function instantiated with the suitable pixel-adapter class.

As the template member-function is instantiated, the appropriate arithmetic operator implementation is used, as specified in the pixel-representation adapter classes. We use the same approach when individual pixels must be rendered as well:

```
// Draw a single pixel into a quad reference
//
    template <class QuadType>
    void DrawPixel(SInt32 xFrame, SInt32 yFrame, QuadType &destination)
    {
    // Draw the pixel if it hits the frame
    //
        if(xFrame >= mFrameRect.right || xFrame < mFrameRect.left ||
           yFrame >= mFrameRect.bottom || yFrame < mFrameRect.top) return;

    // Allow for scaling
    // The drawing parameters are set in LoadImageData()
    //
        const SInt32 yi = static_cast<SInt32>((yFrame - mFrameRect.top) * mYscale) * \
                                                                          mWidth;
        const SInt32 xi = (xFrame - mFrameRect.left) * mXscale;
        RGBAQuad     &picture = mPicture[xi + yi];

        if(PNG_OpaqueAlpha == picture.a) {

        // The picture alpha is opaque, so no blend with its background is required.
        //
            destination = picture;

        } else if (PNG_TransparentAlpha != picture.a) {

        // The picture alpha is transparent, so we must blend with its background.
        //
            const double pictureAlpha = picture.a / 255.0;
            QuadType     macPicture(picture);

        // Blend the background. EXPAND the expression so it gets calculated correctly
        //
            destination = macPicture * pictureAlpha - destination * pictureAlpha + \
                                                                    destination;
        }
    }
```

In this case, the appropriate instantiation is performed automatically, based on the type of the **`destination`** parameter.

5.7.6 Builder — Usage Examples

It is interesting to note that the way we have chosen to use the *builder* design pattern [127] (pp. 97) in the AudioGraph recorder and the players exposes the close similarities between the *template method* [127] (pp. 325) and *strategy* [127] (pp. 315) design patterns. Our approach realises the *template method* design pattern at the top level of the streaming mechanism since the file builder abstract base class provides the means to orchestrate the sequence of builder method invocations. The *strategy* design pattern could be used to accomplish the same feat, but it is generally more useful in a less structured setting, to provide easily interchangeable algorithms. In this particular context, *strategy* would have been less appropriate, as it would have lacked the clear semantic structure specified by the pure virtual methods of **`CFileBuilder`**.

It would also be possible to use the *visitor* [127] (pp. 331) design pattern to achieve similar results to the *builder* design pattern for saving or exporting documents. However, when building documents, the *builder* design pattern is more suitable, as the structural hierarchy to be visited is not yet available and has to be built, which would leave the *visitor* with no structure to visit in such circumstances.

5.7.7 Singleton — Usage Examples

We have used the *singleton* design pattern [127] (pp. 127) in various instances throughout the AudioGraph suite of applications. In the AudioGraph recorder, the application instance is available as a singleton, so that application features are easily available from any client code. Similarly, the preferences manager — which is used for streaming preferences — and the *Window* menu instance — where an up-to-date list of the currently open windows is kept — are available as *singletons*, and are generic to the application.

Asynchronous sound playback services are provided as a *singleton* service in version 1.0 of the recorder and players, as they were provided using services of the Macintosh Sound Manager [157] rather than QuickTime [229]. The initial implementation restricted the number of possible concurrent sound annotations being played to one. The singleton could have been modified to support multiple asynchronous sounds simultaneously, but the effort did not seem worth the trouble at the time.

Version 2.0 of the plug-in uses QuickTime [229] services for sound reproduction, which transparently support multiple concurrent independent sounds, lifting the restriction. It is highly likely that future versions of the Macintosh recorder will use the same approach.

5.7.8 Decorator — Usage Examples

In the AudioGraph system, we have used the *attachment* mechanism in the AudioGraph recorder to handle the processing of mouse events within the episode window. In this manner, we responded by initiating actions as required by the currently selected annotation tools.

Although we have used a single attachment to provide the functionality for the entire set of tools, in retrospect, it may have been more elegant and perhaps slightly more efficient to adopt a variation of the *strategy* design pattern [127] (pp. 315) approach and customise the *attachment* to the specific tool currently selected.

We have also used the *attachment* mechanism to indicate the current top window in the *Window* menu instance, by attaching a `CWindowMenuAttachment` to the menu, and having the *attachment* place a mark against the top window menu item before the menu is expanded.

5.7.9 Command — Usage Examples

The *command* design pattern [127] (pp. 233) has been most useful to us in implementing the multiple undo functionality in the AudioGraph recorder. The action abstract base class provides an interface that requires descendants to implement *redo* and *undo* methods.

Then, the various action classes required to support the authoring tools, or the editing functionality, can take appropriate action when called upon to undo their specific effect or apply it. For instance:

```
//-----------------------------------------------------------------------------
// RedoSelf()
//-----------------------------------------------------------------------------

void CAuthoringAction::RedoSelf()
{
// Insert the component in the Lecture component's authoring component list
//
    mLocation = GetHost()->GetParent()->InsertTopComponentAt( mComponent, mLocation );

// Dirty the host
//
    GetHost()->SetDirty( true );

// Mark installation
//
    mComponentInstalled = true;
}

//-----------------------------------------------------------------------------
// UndoSelf()
//-----------------------------------------------------------------------------

void CAuthoringAction::UndoSelf()
{
// Remove the component from the Lecture component's authoring component list
//
    CAuthoringComponent *removed = GetHost()->GetParent()->RemoveTopComponentAt(mLocation);
    Assert_( removed == mComponent );

// Mark uninstallation
//
    mComponentInstalled = false;
}
```

The redo functionality is executed as soon as the action is created and posted to the undo stack. If the action is to be undone, it is popped off the *undo* stack, the undo functionality is invoked and the action is pushed onto the *redo* stack.

5.7.10 Observer — Usage Examples

We have used the *observer* design pattern [127] (pp. 293) most extensively for handling notifications form controls in the user interface. All controls are *broadcasters*. Virtually all windows that contain controls are also *listeners*, and as controls are instantiated on them, they can be automatically registered with the window *listener*.

As suggested in Section 5.4.2.3.5, we could also use this design pattern to alleviate the dependencies on the edit console window, and assign the responsibility for playback and

selection services in the episode class, without loss of functionality in the edit console window. Any episode instance could then notify its *listeners* of changes in the current selection status. If the associated edit console window exists, it would register itself as a *listener* with the episode. If it is not required, then the episode could function in its absence just as well.

5.7.11 Other Useful Techniques

During testing of the AudioGraph browser plug-in, it is often useful to have the capability to simulate low-bandwidth conditions. Testing the plug-in typically involves viewing a regression test suite of presentations prepared to exercise as much as possible of the capabilities of the tool. Since it is most efficient, convenient and economical to perform such testing without resorting to network access, data is very quickly loaded by the plug-in code, as the bandwidth available between disk and memory is very high.

Therefore, in order to accomplish the desired goal, we devised a mechanism to slow down the influx of data into the plug-in data structures, regardless of the data source, be it local or remote.

The Netscape plug-in framework recommends the use of a class derived from **`CNetscapeStream`** to handle the incoming data stream. **`CNetscapeStream`** provides a virtual method:

```
virtual int32      Write( int32 offset, int32 len, void *buffer );
```

The browser calls this method repeatedly, to supply data to the plug-in as it is received from the data source. The method must return the number of bytes successfully processed by the plug-in.

This arrangement provides us with the opportunity to report that the plug-in has read only one byte at a time, or even report zero bytes every so often, to slow the influx of data even more. We may use:

```
long
CAudioGraphStream::Write( int32 offset, int32 len, void *buffer ) {
#if AudioGraph_USE_SLOW_NETWORK
    static int rollover = 0;
    if(++rollover < 5) return 0;
    rollover = 0;

    return mDestination->WriteData( buffer, 1 );
#else
    return mDestination->WriteData( buffer, len );
#endif
}
```

Rather than simply using:

```
long
CAudioGraphStream::Write( int32 offset, int32 len, void *buffer ) {
    SInt32 retLength = mDestination->WriteData( buffer, len );
    return retLength;
}
```

Notice the rollover static variable used to slow the data stream even further by processing one byte in every five calls to the `Write()` method. This approach can be successfully used to simulate very slow network conditions, and it has proven invaluable in ensuring that the plug-in works as expected in most network bandwidth regimes.

5.8 Summary

In this chapter, we have explored in more detail the technical architecture of the AudioGraph suite of applications. We have reviewed the software architecture of version 1.0 of the recorder application, which the author has designed and built.

We have then discussed the software architecture of versions 1.0 and 2.0 or the web browser plug-ins, which the author has also designed and developed. The software architecture of the standalone version 1.0 player application, also one of the author's contributions, was briefly reviewed as well.

In conclusion, we have explored a number of examples of design patterns in use, which had most relevance to the AudioGraph system implementation.

In the following chapter, we will present the conceptual and technical conclusions we have reached as our research effort drew to a close.

6 Conclusions

In this chapter we critically evaluate our research and present our conclusions.

We assess our progress against our research objectives; show how the AudioGraph has already been used in the field and present insights from a number of informal evaluations that were conducted.

We then discuss our results from conceptual and technical perspectives, and review the effectiveness of our iterative software engineering methodology.

6.1 Evaluation of the AudioGraph System

Our primary research objective has been to provide lecturers with an effective and easy to use system that would assist web-based teaching and learning, allowing them to focus on the application of relevant educational principles rather than requiring them to master arcane technical complexities.

As already suggested in Section 1.3.1 and published in [6], the AudioGraph may have application in all forms of teaching and training, and already it has been demonstrated in at least the following contexts:

- Formal university lectures served on the web [10], [11], [12], [13], [14], [18], [21], [22];
- Formal university lectures provided on a CD ROM [5], [10], [14];
- Tutorials demonstrating worked examples [14];
- Program documentation [3], [4];
- Short promotional web presentations [4].

In addition to these applications, interest has been expressed by a number of companies in using the AudioGraph for producing product-training material, either on CD or published on the web. There are also possibilities in all levels of the educational arena, from primary to tertiary, for reference material to be produced simply and efficiently using the AudioGraph.

Given the high bandwidth required for a good quality video experience, including video data in our presentations would have conflicted with our primary requirements for high quality and low-bandwidth. Although video compression and streaming technologies have advanced considerably in recent years, the situation has remained fundamentally unchanged.

This is not to say that in a few years, when cheap high-bandwidth Internet access may eventually become the norm, video data could not be included if so desired. However, video data production is likely to continue to be more difficult than authoring slides and annotating them, which might hamper the ease of use.

The AudioGraph has been used for presenting formal university lectures both at Massey University and at Surrey University. In fact, Pearson and Jesshope [5] presented the course at Massey University in cross-campus mode, in collaboration with Waikato University. With the small numbers of students involved in the joint Massey/Waikato course we could do no more than an informal evaluation of the AudioGraphic mode of lecture presentation.

Although a more formal evaluation would have been preferable in order to gain a better appreciation of the various strengths and weaknesses, even the informal evaluation proved to be very fruitful, as several issues concerning the quality of the recording were raised that given their technical nature could later be easily overcome.

Also, it was most interesting to note that the student grade point averages had increased markedly over the previous year when the AudioGraph was not used, even though the lecturer and the underlying content of the presented material did not change, as also reported by Jesshope in the context of a postgraduate paper on advanced computer systems [14].

Indeed, from the lecturers' perspective we have had unconditional praise for the concept from those who have used the software as well as numerous suggestions for improvements to the software, which have been incorporated already into subsequent versions of the software.

6.2 Conceptual Aspects

We also sought to fulfil our primary research objective by building on the conceptual and technical requirements arising from the AudioGraph idea and its prototype implementation, supported by evaluation of other existing systems and approaches.

In order to provide an intuitive metaphor for our target users, lecturers and students, we designed the AudioGraph web-based lecturing system around the concept of *lecture*, which would contain a number of *episodes*. An *episode* would be considered the equivalent of a transparency slide.

Just as slides can be chosen and stacked in the right order prior to lecture delivery, we designed the system so that the choice and sequence of *episodes* in a *lecture* can be organised through simple drag-and-drop or cut and paste operations, facilitating lecture preparation.

When preparing for web-based delivery, the entire *lecture*, or a specific user-defined selection of *episodes* from the *lecture*, can be exported in web-ready format, eliminating the need to edit any HTML [190] files, since their sequence can be correctly defined in advance.

In order to provide lecturers with an equivalent experience for the transparency film that can be scrolled over the projector, we have made available two types of *episode*: the *slide* and the *scroll*. *Slides* have a defined width and height, which governs the size of the web browser plug-

in display area, which is immutable. *Scrolls*, on the other hand, have a display area of virtually unlimited height[1], over which the current view can be scrolled, replicating the scrolling transparency experience.

As we sought to burden our target authors — lecturers — with minimal technical sophistication, we have retained the prototype's concept of a single presentation stream, with sequenced audio and graphical annotations and no concurrency requirements, as discussed in [1].

In order to give good feedback about the types of annotations present in the recorded stream, we have adopted an *iconic* representation, as later categorised by Siglar [48], which could reflect annotation types and their associated attributes, as can be seen in Figure 3.9.

Using the edit console, annotations can be selected, moved on screen or re-arranged in the presentation stream as desired, facilitating an efficient editing experience.

So that groups of graphical annotations could be displayed at once, providing a sharp switch between animation frames, and therefore good quality graphics animation, we made the *stealth* graphical annotations attribute available. This attribute causes the graphical annotations' display effect to be delayed until the first non-stealthy annotation is encountered in the playback stream, leading to smooth frame animation.

Version 2.0 of the web-ready file format extended the definition of the *highlight group* concept, which mirrors, to a certain extent, the concept of *stealth*.

In this new approach, for annotations with the highlight attribute, the player accumulates its effect without updating the screen. The accumulated effect then modifies the way in which the screen is displayed — according to the normal highlighting display mode — only when pause, audio or other non-stealthy and non-highlighting annotations are encountered. The accumulated *highlight group* continues to modify the way in which the screen is displayed until either a new *highlight group* is defined or until a special 'end of highlight' annotation is encountered. In this manner, at any time, the current highlight group, if one exists, transforms the way in which the screen is displayed, in the area defined by the highlight components.

All AudioGraph player applications are capable of *streaming* playback, meaning that the presentation can begin as soon as an annotation for visual display or aural playback has been received, while the rest of the presentation is still downloading. Since it is not necessary to wait for the episode's entire web-ready file to be downloaded before playback starts, the system appears more responsive and the user experience is improved.

[1] The coordinate system uses 32-bit integers, which can be used to represent a few billions of pixels, being therefore more than enough for all practical uses.

From a conceptual perspective, there is a perpetual tension between the structure and appearance of information. Based on a particular viewpoint, and the specific value system used, various different approaches to creating and structuring information can be taken.

For instance, some document editors seek WYSIWYG capability — such as Microsoft Word [275] — that may induce users to focus more on the look-and-feel of their work rather than on ensuring a sound semantic structure. In this case, precise control over individual representation elements with immediate visual feedback is prized above uniformity of presentation and strict adherence to clearly defined style guides.

Other solutions — such as LaTeX [292] — focus more on the logical structure of the documents and leave the rendering details as the responsibility of automated tools that will generate correct representations based on precise typographic style standards. In this approach, the ability to focus on semantics and logical structure unencumbered by visual distractions and the standardised presentation layout based on predefined styles are valued above immediate graphical feedback. The open source Apache Cocoon [293] project takes a similar approach, separating content, style, logic and management functions in XML [167] content-based web sites and web services.

Lecturers may accumulate considerable amounts of educational materials organised in formats that are not necessarily directly suitable for web-based lecturing, such as PowerPoint [64] slides, text documents, diagrams, graphics, etc. We observed that in order to be truly effective, rather than requiring lecturers to author semantically identical material from scratch into a new form, a web-based lecturing system should be capable of absorbing and reshaping existing material.

The early AudioGraph prototype — prior to this research — illustrated a new media concept, which brought together a background image, graphical annotations and voice clips, and presented them in a single dynamic stream, as discussed in [1]. It had no specific capabilities and placed no emphasis on reuse of previously existing material in any other forms apart from background images.

The AudioGraph seeks a medium between the approaches discussed above by having the ability to absorb ready-made presentations — whose structure we assume to be sound already — and providing WYSIWYG editing capabilities for ease of extension and multimedia annotation.

The author's AudioGraph work started from the premise of existing presentation material — presentation slides and course notes — and sought to facilitate the absorption of such documents into the new media paradigm as easily as possible. That is why version 1.0 of AudioGraph recorder, designed and developed by the author, can import PowerPoint [64] presentations and arbitrary background image collections. The author's document reshaping idea went even

further, when Paul Lyons [23] pointed out that he has lots of Word documents rather than PowerPoint [64] slides — although this capability has not yet been implemented.

The author then offered the idea that we could envisage an editing workspace into which the text documents can be imported, and from that point on we can proceed to partition the presentation into suitable sections that could be regarded as voice clips. In other words, the text could be seen to form the transcript for the voice patches — or at least a good starting point for them — and from there it was a simple jump to suggest text-to-speech synthesis for a first cut of the presentation.

Another crucial contribution has been the episode and generic URL [196] linking capability, as captured in version 2.0 of the web-ready file format and realised through a unified, consistent access scheme, with flexible capabilities. Various other new elements were also introduced in the AudioGraph specification to support this capability, such as the 'stop', 'clearscreen' and *persistent* image annotations.

The 'stop' annotation would halt the presentation stream. As hyperlinks are not active unless the playback is stopped, this annotation effectively provides a means of activating the hyperlinks.

The 'clearscreen' annotations and their relationship with *persistent* image annotations are described in more detail in Section 3.3.4.6.

Related to the previous discussion of information structuring capabilities, this presentation linking approach makes it possible to define the collections of learning paths through lectures as full graphs rather than mere hierarchical trees. From multiple perspectives, it has gradually become apparent that flexible learning paths are valuable for effective web-based education outcomes, as shown by Weippl [41], Iiyoshi and Hannafin [42], Parsley [43] and Raphael [44].

6.3 Technical Aspects

To summarise, the main practical goals we have sought to achieve were ease of use throughout the system, good quality voice, image and graphical annotations and low bandwidth requirements for streaming the AudioGraph system's educational content output.

Version 1.0 of the AudioGraph recorder provides tools for drawing straight lines, rectangles, arcs and ovals. Also, it provides convenient means of creating filled graphics, such as filled rectangles or ovals, by offering specialised tools for this purpose, just like most other graphics editing packages.

For all its images and graphical annotations, the AudioGraph supports up to 256 levels of transparency — between fully opaque and fully transparent. We consider that fine-grained

control over annotation transparency and the erasing effect can facilitate expressiveness for advanced users.

The colour, pen size, transparency, stealth, highlight and erasing attributes are editable for each graphical annotation, at any point in time, using the controls shown in Figure 3.12.

The AudioGraph system offers the flexibility of choosing from the full palette of millions of colours available in a 24-bit RGB colour space.

From version 1.0 of the AudioGraph onwards, all episode *components* can be annotated with text comments. For audio patches, these comments can be used to store the transcript for audio patches. In the future, such transcripts might even be used as the source for computer-synthesised speech as an alternative to recorded human speech.

In version 1.0 of the AudioGraph, although we kept the concept of a background image, multiple images are available in *episodes* for display or animation. It is possible to display and hide images at will, zoom images in or out, crop or stretch them to fit given rectangles.

We initially assumed that most images would contain text and relatively simple graphics, being potentially derived from educational PowerPoint [64] slides. It was deemed important that the text, at all font sizes, should be faithfully represented, without any compression-induced artefacts[2], and with the greatest bandwidth economy possible. Also, the relatively monochrome nature of typical lecture graphics and text — as opposed to the full-colour gradients present in photographic images — reinforced our conclusion that loss-less image compression formats — such as PNG [220] — could best satisfy this set of desirable qualities, as explored in more detail in Section 4.2.1. With the adoption of PNG [220], images can also support transparency.

Although the author envisaged text annotations, to support a crisper expression than hand-drawn text, it was not until version 2.0 that they became fully implemented.

As most lectures consist of voice-annotated graphics animations, we sought a compression format optimised for voice, both economical in terms of bandwidth and providing excellent quality, with as little distortion as possible. As more comprehensively discussed in Section 4.2.2, we found that GSM 06.10 [227] satisfied these criteria well.

To facilitate ease of editing for episodes with large numbers of annotations, we created an annotation grouping mechanism, as discussed in Section 3.5.2.2.

True to our requirements, and consistent with good human interface design principles, as discussed in Section 4.1.1.8, version 1.0 of the AudioGraph implemented multiple undo and redo functionality.

---

[2] As can be seen in Figure 4.5.

Version 2.0 of the web-ready file format introduced the option of requiring the player plug-in to display without its controls or border. Playback can still be stopped and started by clicking in the display rectangle. Episodes that take advantage of this option blend into their surrounding web page without showing any border or playback controls, much like a banner advertisement.

In the process of creating the user interface designs for the Macintosh recorder, the player and browser plug-in, the author has gained deep insight into the technical underpinnings required to support a simple interface. To this end, a critical design decision was the choice of iconic editing for annotation management.

It is worthy of note that all the core user interface design choices have been propagated to all the current applications on both the Macintosh and the PC — Version 2.0 of the Mac recorder and plug-in and the PC recorder. However, the user interface controls of the PC plug-in have yet to be implemented to mirror the Macintosh implementation.

Although a different contributor implemented text annotations in version 2.0 of the AudioGraph recorder, the author envisaged the use of anti-aliasing for high-quality text display and the efficient use of PNG [220] for storing platform-independent text graphics by choosing appropriate colour palette structures.

The author also argued for the ability to use text components in the foreground layer if desired, which would have meant that it would have been very easy and efficient to erase or animate them when required. However, this capability has not yet been included in the web-ready file format specification.

From an engineering perspective, the following contributions are worthy of mention:

- Efficient rendering — The complete rendering engine overhaul for Version 2.0 of the plug-in that led to `O(1)` playback characteristics for virtually all presentations — in keeping with the *make the frequent cases fast and the rare case correct* motto ([141] pp. 126). This effort also had input into the precise formulation of the specification document for version 2.0 of the file format, as certain specification choices would have precluded the possibility of developing an `O(1)` rendering engine.
- Pixel-depth-independent rendering — A creative application of the *strategy* design pattern [127] (pp. 315), which led to an elegant and efficient implementation.
- Region-handling optimisations — A device for speeding up region traversals by taking advantage of byte comparisons rather than using bitwise operators at all times, also in keeping with the *make the frequent cases fast and the rare case correct* principle ([141] pp. 126).

6.4 Advantages of an Iterative Software Engineering Methodology

Our third main research objective dealt with the application of sound human computer interface design and software engineering principles in the design and development of a web-based lecturing system. We believe we have successfully met this objective in light of the simple, elegant and guideline-compliant interface designs and the robust and efficient software components we have produced.

From a software engineering perspective, for the design and development of the AudioGraph system, we have used an iterative, incremental software construction methodology, similar to the recommended approach popularised by the Unified Process [146]. Our iterations contained the following activities:

- Requirements gathering and analysis. Where appropriate, in order to supplement and clarify requirements, the author used mock-ups to sketch the user interface vision and focus the construction efforts in the areas of greatest risk first. This approach also helped us test human interface design decisions early in the process, while maintaining adherence to the human interface guidelines.
- Design. The design efforts continuously focused on executable architecture. Rather than exhaustive design up-front, the author prepared sufficient design artefacts to support the delivery of executable software in short iterations — from a few days to a couple of weeks. This approach ensured that the software grew continuously through executable forms rather than passing through longer periods of uncertainty.
- Construction. In each of the iterations, either a set of new features was built, or the architecture of the system was re-factored [205] to improve quality and performance.
- Testing. We did not have the benefit of an automated testing framework, and had to rely primarily on manual testing effort. However, for the AudioGraph player plug-ins, regression-testing suites were built, for both versions 1.0 and 2.0 of the AudioGraph web-ready file format. These suites contained episodes that exercised all the features of the AudioGraph playback engine, and they proved invaluable in ensuring that in the process of refining the rendering engine and creating performance enhancements the correct playback behaviour was preserved at all times.
- Evaluation. After each of the iterations, we evaluated the progress to date and planned the effort for the next iteration.

This lightweight, iterative approach to software construction has ensured that we were able to create valuable software, of very good quality, in relatively short timeframes. As a result, the software could be put to use in real-life settings very rapidly, and we could readily test the fruits

of our research. For instance, Pearson and Jesshope [5] presented a course that took advantage of robust in-progress releases of the AudioGraph system at Massey University in cross-campus mode, in collaboration with Waikato University, and other courses [18] began to use AudioGraph components as well, such as the Foundations of Computer Science [21] and Networks for Parallel Computers [22] offerings from Massey University.

6.5 Summary

In this chapter we have reviewed the conceptual and technical results we have achieved in light of the objectives of our research, and highlighted our relevant contributions.

We have shown that the AudioGraph system has been effectively used for web-based teaching and learning. We have then described how the AudioGraph system's document absorption and reshaping capability provides a useful approach for reusing and enhancing existing lecture presentation material.

We have found that we have successfully met all of our conceptual and most of our technical objectives, with the exception of text annotations — which were later implemented in version 2.0 of the AudioGraph recorder by other contributors — and support for the Ogg Vorbis sound format [242].

Also, version 2.0 of the web-ready file format brought with it a number of enhancements beyond our original requirements, such as the hyperlinking support, enhanced highlight groups and border-less plug-in display.

In the next chapter, we will highlight promising avenues of future research.

7 Future Work

In this chapter, we highlight a number of future avenues of investigation that show promise.

From a conceptual perspective, there are various evaluation and iterative improvement opportunities that might be pursued in order to provide a sound foundation for refining the AudioGraph system. Approaches for smooth integration in Learning Management Systems should also be investigated, in order to best blend AudioGraphic content within the wider educational context.

From a technical perspective, given the increasing standardisation for web-oriented multimedia integration provided by the Synchronised Multimedia Integration Language (SMIL 1.0 [187] and SMIL 2.0 [188]), the AudioGraph system might be extended support it. Other alternative distribution formats could also be considered, such as Macromedia Flash [88], Apple QuickTime [228] or MPEG-4 [249].

Extended media support may also be investigated in order to benefit from alternative formats and advances in image, sound and video compression. We conclude this chapter with a number of other technical improvement opportunities such as multimedia storage issues.

7.1 Evaluation and Iterative Improvement

One of the major benefits in developing our own system is that it can evolve to serve diverse and potentially unforeseen interests — unrestricted by the technical limitations or release schedules of any other particular system — as guided by research activities in the web-based education and knowledge-management arenas.

More efforts could be spent to formally evaluate the effectiveness of the AudioGraphic lecturing approach, beyond the study of the AudioGraph prototype conducted by Segal in 1997 [2] and the informal studies reported in [5] and [14].

A number of studies — focusing on various aspects — could be conducted to determine the impact the AudioGraph lecturing system has on students and lecturers. For instance, the wider implications for student learning and course design might be considered.

To this end, feedback might be constantly sought from students taking courses with AudioGraphic components. This approach could be used to inform course design improvement activities, in an iterative fashion.

Similarly, lecturers preparing AudioGraphic material may be periodically interviewed in order to better understand their evolving priorities and authoring requirements. We could then test their satisfaction with the current user interface and elicit their suggestions for further

enhancements and user interface improvements. Further research could also focus on developing specific guidelines for AudioGraphic content preparation in alignment with modern instructional design principles, as discussed in Section 1.2.

Much research has been done in the area of Computer Support for Collaborative Learning (CSCL [63]). Jonassen et al. [60] argue that currently the challenge lies in providing *supportive* rather than *intervening* learning environments. In other words, in certain contexts, rather than focusing students primarily on the acquisition of established knowledge, they could engage in making contributions to the collective knowledge, and teaching would become the activity of supporting knowledge-building communities, as discussed by Scardamalia and Bereiter [61].

In this context, enhanced notes management could be another interesting enhancement aimed at the students using the AudioGraph players to view educational material. Typically, during traditional lectures, students take notes. In the AudioGraph learning context, it might prove useful to enable students to make annotations on-line while they are viewing AudioGraph lectures, and associate the notes with specific locations within the episodes. Then, when reviewing the material, the matching annotations would be available where they are most pertinent and might also be used to focus attention in relevant areas of interest.

There could be a number of alternatives for providing this kind functionality, either standalone — in local or centralised storage — or integrated into a larger Learning Management System. Such integration could ensure that all notes are stored into a central knowledgebase, and students would be able to access them regardless of which workstation they would use to connect to the system.

Further research effort might also be spent into investigating whether the AudioGraph could be extended effectively to support quizzes and tests, or how it might be integrated with such verification elements created with other specialised authoring tools.

In addition, although the AudioGraph web-ready file format makes provisions for the synchronous delivery of presentations, it has not yet been implemented. Developing this capability would provide an alternative to the current asynchronous delivery paradigm, and might enhance the appeal of the system for some users seeking more flexibility in using the AudioGraph system.

The W3C [163] has published in May 1999 a Recommendation for Web Content Accessibility Guidelines [181], aiming to explain how to make web content accessible to people with disabilities, and promote accessibility. A further Working Draft for version 2.0 of the WCAG [182] has been put forward in June 2003. Incorporating feedback on WCAG 1.0 [181], it focuses on core and extended checkpoints for each of the four main guidelines — whose keywords are: *perceivable*, *operable*, *understandable* and *robust*. It attempts to apply

checkpoints to a wider range of technologies and to use wording that may be understood by a more varied audience.

In this context, the AudioGraph system could be reviewed in light of these guidelines, and suitable improvements could be designed. For instance, we might perhaps wish to explore the idea of providing a captioning mechanism for AudioGraphic presentations. As an example of another specific technical initiative in this arena, an excellent tool [271] has been developed to support Macromedia Flash [86] captioning [272], making Flash [86] captioning practical.

The tool is an ActionScript [90] — the Flash [86] scripting language — component that parses a MAGpie [273] captioning XML file and displays the caption data within a Flash [86] presentation. MAGpie [273] is a free, de-facto standard multimedia captioning application. In the same spirit, possibly based on the availability of sound transcripts[1], interesting research could be done to explore ways in which the AudioGraph system might generate captions.

7.2 Integration in Learning Management Systems

As shown by Jesshope, Heinrich and Kinshuk [16], Jesshope [17] and Heinrich, Jesshope and Walker [20] efforts have already begun to integrate the educational content created by the AudioGraph system in advanced Learning Management Systems. For instance, very interesting work has been undertaken to provide novel means of searching for specific information using a query language close to natural English [20].

To enhance acceptance of the AudioGraph system in modern educational institutions, and increase its integration capabilities into other Learning Management Systems, it is very likely that AICC [122] and SCORM [124] standards compliance will need to be pursued.

The term *AICC Compliant* [123] means that a training product complies with one or more of the nine AICC Guidelines & Recommendations (AGR's). Since there are nine different AGR's, *AICC compliance* may have different meanings, as there are nine different compliance options.

The AICC — the US Aviation Industry CBT (Computer Based Training) Committee — has developed formal certification testing procedures for the Computer Managed Instruction (CMI) related AGR's (AGR-006/AGR-010) and currently offers certification testing for both CMI systems and CBT courseware.

By adapting, as required, the various components of the AudioGraph web-based lecturing system to be fully AICC compliant, we could ensure our solution meets current best industry

[1] Version 1.0 of the AudioGraph recorder already provided the ability to record a textual transcript with sound annotations, as shown in Figure 4.4.

practices, and it would continue to provide a sound platform for developing and delivering web-based lecture material.

The Sharable Content Object Reference Model (SCORM) [124] defines a Web-based learning *Content Aggregation Model* and *Run-Time Environment* for learning objects. The SCORM is a collection of specifications adapted from multiple sources to provide a comprehensive suite of e-learning capabilities that enable interoperability, accessibility and reuse of Web-based educational content. The work of the Advanced Distributed Learning [125] initiative to develop the SCORM is also a process to knit together disparate groups and interests. This reference model aims to coordinate emerging technologies, commercial and public implementations. The SCORM includes aspects that affect Learning Management Systems and content authoring application vendors, instructional designers, content developers and training providers.

SCORM [124] consists of three main sections:

1. An XML-based [169] specification for representing course structures that facilitates course migration between Learning Management Systems or servers.
2. A set of specifications relating to the run-time environment, including an API, content-to-LMS data model, and a content-launch specification.
3. A specification for creating meta-data records for courses, content, and raw media elements.

These specifications enable the reuse of web-based educational content across multiple environments and products.

By adopting the standards, recommendations and guidelines included in SCORM [124], it will become possible to define all the course elements, structure, and external references in such a manner as to facilitate the easy transition of courses from one Learning Management System environment to another.

7.3 Alternative Distribution Formats

Since the AudioGraph web-ready file format is proprietary, it requires custom players. This limitation, if not addressed, may prove detrimental to the widespread adoption of the AudioGraph web-based lecturing system. Therefore, a number of alternative distribution formats — such as SMIL 1.0 [187] or SMIL 2.0 [188], Macromedia Flash [88], Apple QuickTime [228] or MPEG-4 [249] — might be considered as options for web-ready representations of AudioGraphic educational material.

7.3.1 Synchronised Multimedia Integration Language (SMIL)

Given the understandable rise in popularity enjoyed by XML [167] and its related technologies, it is likely that in the near future efforts may be undertaken to provide the ability to export AudioGraph presentations into SMIL 2.0 [188], even though much of its support for concurrency in presentations is likely to remain unused in this context. For the graphical annotations, it is to be expected that the SVG [186] XML application will be used, which is briefly described in Section C.2.

It would be possible to continue to present image data in PNG [220] format. However, as SMIL [188] offers the ability to present alternative representations depending on the capabilities of particular player or platform, alternative representations might be provided as well.

This feature may be useful in providing viable alternatives for representing sound and image annotations so that a wide variety of clients can experience the AudioGraphic presentations without disruption.

The Synchronised Multimedia Integration Language (SMIL 1.0 [187] and SMIL 2.0 [188]) — pronounced *smile* — is an XML application that facilitates the integration of sets of independent multimedia objects into synchronised multimedia presentations.

Using SMIL [188], authors can:

- Describe the temporal behaviour of the presentation.
- Describe the layout of the presentation on a screen.
- Associate hyperlinks with media objects.

XHTML 1.0 [192] is a reformulation of HTML [191] into a language that conforms to the XML 1.0 Recommendation [169]. The ultimate goal of this reformulation is that XHTML and its descendants [193] should be useful in environments where there are no preconceived notions about the semantics of any element or attribute — generic, adaptive XML environments.

Although the actual realization of this goal lies still in the future, efforts are already being made at standardization committee level to ensure that documents developed using XHTML will be portable into these adaptive environments even if currently the documents must be processed by *user agents* with specific knowledge of some of the document elements and attributes (e.g. <form>, <applet>).

On the path toward such adaptive XML environments, modularisation is a key step. Widespread growing demand exists for capabilities to extend HTML [191] in various ways, in order to accommodate device-specific functionality, to limit the content that is sent to smaller-footprint devices, or to enhance the ability to produce useful Internet content. In order to satisfy such diverse requirements, a framework is being defined for developing markup languages derived

from HTML [190]. Once in place, this framework would be used as a means for defining extensions to XHTML 1.1 [193], and as a set of building blocks that markup language designers could use to bring the extensions together with the base into a cohesive whole.

As part of this modularisation effort surrounding XHTML [195], version 2.0 of SMIL [188] strives to allow reuse of SMIL syntax and semantics in other XML-based languages, in particular those who need to represent timing and synchronization. For example, SMIL 2.0 components can be used for integrating timing capabilities into XHTML 1.0 [192] and 2.0 [193] and into SVG [186]. Appendix C gives a wider perspective on XML, its evolution and its related technologies relevant to our research.

7.3.2 Flash

To take advantage of its popularity, it is conceivable that the AudioGraph presentations might be exported into the Flash file format [88], since it is freely available and vast numbers of users already have available suitable Flash player plug-ins [87] for their browsers. This file format has already started to become adopted for emerging hardware platforms such as the Nokia 9200 Communicator [89]. It is highly likely that this trend will continue, and multimedia support for mobile devices and PDA's will increase in the future, as will their capabilities.

Playing back a Flash file involves rendering a sequence of frames [88]. Displaying one such frame is a three-stage process:

1. Objects are defined, and each of them is given a unique identifier, called a *character*, and stored in a repository called the *dictionary*.
2. Selected *characters* are copied from the *dictionary* and placed on the *display list*. This is the list of the *characters* that will be displayed in the next frame.
3. Once fully defined, the contents of the *display list* are rendered to the screen by using a special command.

Each *character* in the *display list* is assigned a *depth* value. The *depth* determines the stacking order of the *character*. *Characters* with lower depth values are displayed underneath *characters* with higher depth values. *Characters* may also appear more than once in the *display list*, at different depths. There can be only one *character* at any given depth.

In earlier Flash format definitions, the *display list* was a flat list of the objects that were present on the screen at any given point in time. In the current format definitions, the *display list* is a hierarchical list where an element on the display can have a list of child elements.

The graphical annotations can be defined using the shapes provided by the Flash file format. The fact that Flash supports transparency ensures that transparent graphical annotations can be mapped to successfully.

Image annotations can be implemented using sprites. Vanishing images would probably be implemented by removing the appropriate image annotation from the Flash *display list*. It is important to note that Flash [88] does not support PNG [218] image data natively. Instead, it supports JPEG [216] and ***zlib***-compressed [224] raw binary bitmap data. As a result, AudioGraph images would have to be trans-coded to either JPEG [216] or Flash [88] bitmaps.

Flash supports sounds with sample rates of 5.5, 11, 22 and 44 kHz in both stereo and mono [88]. It is assumed that the playback platform can support sampling rate conversion and multi-channel mixing of these sounds. The number of simultaneous channels supported depends on the capabilities of specific platforms, but is typically in the range of three to eight channels. From the AudioGraph perspective, only the streaming sounds are of interest. The event sounds, associated with control clicks are not so relevant.

Streaming sounds are downloaded and played in tight synchronization with the timeline. In this mode, sound packets are stored with each frame. If a particular computer cannot render the frames of a movie as quickly as specified by the frame rate of the movie, the player slows down the animation rate instead of skipping frames, which is exactly the behaviour we are after.

Flash supports Adaptive Differential Pulse Code Modulation (ADPCM [235]) and MPEG-1 Layer 3 (more commonly known as MP3 [252]), and later versions of MPEG-2 [246] that were designed to support lower bit-rates.

ADPCM [235] is a family of audio compression and decompression algorithms. It is a simple but efficient compression scheme that avoids any licensing or patent issues that arise with more sophisticated sound compression schemes, and helps to keep player implementations small.

ADPCM [235] uses a modified Differential Pulse Code Modulation (DPCM) sampling technique where the encoding of each sample is derived by calculating a *difference* value, and applying to it a complex formula that includes the previous sample values. The result is a compressed code, which can recreate nearly the same subjective audio quality.

A common implementation takes 16-bit linear PCM [234] samples and converts them to 4-bit codes, yielding a compression rate of 4:1. The Flash implementation supports 2, 3, 4 and 5-bit ADPCM [235] codes.

Given these capabilities, GSM 06.10 [227] sound captured by the AudioGraph would need to be trans-coded into either ADPCM [235] — license free — or a variant of MP3 [252] — with licenses payable to Thomson Multimedia and Fraunhofer [253].

Care would need to be taken in replicating the special behaviour of highlight groups as defined by the AudioGraph file format, as new highlights trigger the disappearance of all highlights in the previous highlight group.

As current versions of the Flash file format [88] provide hyper-linking capabilities, and allow multiple movies to be played in the same Flash player [87], the matching AudioGraph functionality could be mapped into the Flash file format [88].

One interesting issue to address when exporting AudioGraph lectures into the Flash file format [88] would be the provision of equivalent presentation controls. The version 2.0 of the AudioGraph web-ready file format specification provides for the ability to display presentations without any controls, in which case the equivalent Flash file would be relatively straightforward to generate.

However, if we wish to provide a fully equivalent user experience using the Flash file format [88], then the AudioGraph player component controls, as illustrated in Section 3.6, would need to be implemented as part of the presentation itself, since the Flash player [87] does not offer any predefined controls. It is very likely that the extensive scripting capabilities of Flash will provide the answer to this problem.

Given the large number of features provided by the Flash (SWF) file format [88], designing an efficient mapping of the AudioGraph annotations into Flash tags in order to provide an equivalent user experience could prove to be an interesting research challenge.

7.3.3 QuickTime

Using similar concepts and approaches as the ones developed for mapping to the Flash (SWF) file format [88], AudioGraphic content could also be exported in the QuickTime file format [229]. It is interesting to note that QuickTime provides support [230] for playing back both SMIL 1.0 [187] and Flash [88] content.

QuickTime provides natively similar capabilities to Flash, and in many instances is much more powerful, supporting large numbers of video and audio formats. QuickTime provides support for vector graphics, sprites and transparency, which means that graphical AudioGraph annotations can be implemented efficiently. QuickTime [228] does support PNG [220] natively, so image data could be copied across from AudioGraph episodes to QuickTime movie files without transcoding.

QuickTime [228] also supports hyperlinking, which means that episode linking functionality could be provided as well.

Although QuickTime supports a wide variety of sound formats, GSM 06.10 [227] is not directly supported by it. Either we could develop GSM 06.10 [227] decoding components for both the Macintosh and Windows platforms, effectively extending QuickTime to support it, or we could transcode AudioGraph sounds into a format natively supported by QuickTime, such as MP3 [252] or AAC [254].

The QuickTime player and web-browser plug-in provide controls that are relatively similar to the ones used by the AudioGraph players. The most significant differences between the two alternatives become noticeable in the presence of multiple episodes presented in the same plug-in instance. Whereas the AudioGraph plug-in indicates the boundaries between episodes, the QuickTime players offer no such feature.

7.3.4 MPEG-4

Early audio-visual standards (circa 1993) established by the Moving Picture Experts Group (MPEG [244]), a working group of ISO/IEC in charge of the development of standards for coded representation of digital audio and video, focused primarily on encoding audio and video bit streams. It produced the MPEG-1 standard [245], on which such products as Video CD and MP3 [252] are based and the MPEG-2 standard [246], on which such products as Digital Television set top boxes and DVD [247] are based.

Marking a major departure form previous audiovisual standards, and forming the basis of significant new capabilities, the audiovisual representation model that underpins MPEG-4 [249] is also object-based, making it another excellent potential candidate as a target for AudioGraph educational content export. MPEG-4 [249] is the new standard for multimedia for the fixed and mobile web and MPEG-7 [250] is an associated standard for the description and search of audio and visual content.

As we have already discussed in Section 4.2, using objects that algorithmically define the visual or audio representation of an audio-visual stream is a particularly efficient method of data compression, although it might not result in perfect reproduction in all cases.

Using this approach — rather than attempting to compress the data stream of the original representation itself — the desired user experience can be recreated with the minimum of data transferred between the communication points. The AudioGraph web-ready file format, Flash [88], Shockwave [91] and certain forms of QuickTime [229] are some other examples of file formats that have taken advantage of this observation.

An object-based scene is built using individual objects that have relationships in space and time, offering a number of significant advantages:

1. The various object types may take advantage of their most advantageous coded representation — a synthetic moving head is clearly best represented using animation parameters, while video benefits from the efficient encoding of pixel values. Voice and music have different characteristics and benefit from tailored encoding schemes.
2. The different types of data may be harmoniously integrated into the same scene: an animated cartoon character in a real world, or a real person in a virtual studio set.
3. Interacting with the objects and hyper-linking from them is now feasible in the context of MPEG-4.

More advantages could be enumerated, such as selective spending of bits, easy re-use of content without trans-coding, providing sophisticated schemas for scalable content on the Internet, and so on.

The Extensible MPEG-4 Textual format (XMT [249]) is a framework for representing MPEG-4 scene descriptions using a textual syntax. The XMT allows the content authors to share their content with other authors, systems or service providers, and facilitates interoperability with both the Synchronised Multimedia Integration Language (SMIL) [188] from the W3C consortium [163], and the Extensible 3D (X3D) [189] being developed by the Web3D Consortium.

The XMT format is interchangeable between SMIL players, VRML [251] players, and MPEG-4 players. The format can be parsed and played directly by a W3C SMIL player, pre-processed to Web3D X3D and played back by a VRML player, or compiled to a MPEG-4 representation such as mp4, which can then be played by an MPEG-4 player.

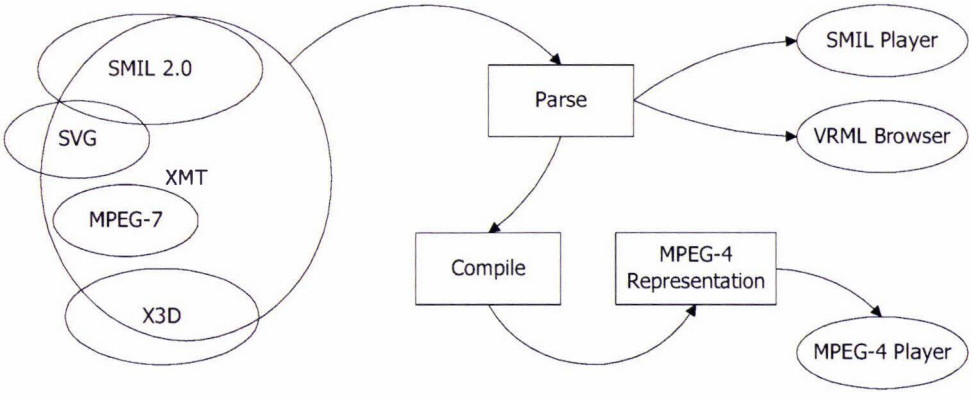


Figure 7.1.The eXtensible MPEG-4 Textual format

Figure 7.1 gives a graphical representation of the interoperability capabilities provided by the XMT format [249]. It encompasses MPEG-4 [249], a large part of SMIL [188], much of the Scalable Vector Graphics [185] and X3D [189] specifications and gives a textual representation for MPEG-7 *Descriptions* [250].

The XMT framework consists of two levels of textual syntax and semantics: the XMT-A format and the XMT-Ω format [249].

The XMT-A is an XML-based version of MPEG-4 content, which contains a subset of the X3D. Also contained in XMT-A is an MPEG-4 extension to the X3D to represent MPEG-4 specific features. The XMT-A provides a straightforward, one-to-one mapping between the textual and binary formats.

The XMT-Ω (also known as XMT-O, since the omega symbol is not quite so widely used on the web) is a high-level abstraction of MPEG-4 features based on the W3C [163] SMIL [187]. The XMT provides a default mapping from Ω to A, for there is no deterministic mapping between the two, and it provides content authors with an escape mechanism from Ω to A.

In conclusion, although the capabilities of MPEG-4 [249] surpass significantly the relatively modest multimedia demands of AudioGraph lectures, it should not be discarded from consideration for future extensions to the AudioGraph system.

7.4 Extended Media Support

As the use of the AudioGraph system continues to increase, it is likely that our initial simplifying assumptions about the media types most likely to be used in AudioGraphic presentations — images with monochrome text and relatively few colours for backgrounds; voice sound clips — are likely to be challenged, as more users will potentially wish to use more photographic images and demand better sound quality.

The AudioGraph file format does not intrinsically restrict the image definitions to PNG format and the recorders and AudioGraph players could easily support other image formats, such as JPEG [216] for instance for photographic images. This could be achieved either through the use of QuickTime [229] or by using custom imaging libraries that support JPEG, perhaps such as the one provided by the Independent JPEG Group [217].

Using QuickTime would require the presence of QuickTime on the target platform, while using the custom library would free us from such limitations. It is conceivable that our code could support both options, taking advantage of the power and flexibility of QuickTime if it is available, and resorting to the supporting library code if it is not.

In fact, version 1.0 of the AudioGraph for Macintosh browser plug-in used to take precisely this approach, by using QuickTime if it was available and falling back to using the *libpng* [218] library if it was not. By using that particular approach, that version of the plug-in could render any image format supported by QuickTime.

However, the inherent performance limitations of that rendering engine guided us toward a solution requiring in-memory RGBA decoding of the image data rather than rendering it directly to the screen. This shift in requirements made the *libpng* approach more attractive, and version 2.0 of the AudioGraph plug-in uses it exclusively for the time being.

Since JPEG [216] images do not support transparencies natively, we could support them quite easily in the 2.0 plug-in rendering framework by assuming that any such pictures are fully opaque.

In terms of sound compression, GSM 06.10 [227] gives good quality results and has relatively modest bandwidth requirements. However, its licensing status is questionable, and other more exciting alternatives may become available.

One such candidate may be the Ogg Vorbis [242] file format, which aims to support mid to high quality — 8kHz to 48.0kHz, 16 bit and above, polyphonic — audio and music at fixed and variable bit-rates from 16kbps to 128 kbps/channel.

Although not specifically tuned for voice, this format promises substantially better sound quality, with tuneable bandwidth requirements. Best of all, this format is patent and royalty free, and in terms of quality sits in the same competitive class of audio representations with MPEG-4 AAC [254] (Advanced Audio Coding), and is similar to, but offering higher performance than MPEG-1/2 audio Layer 3 (better known as MP3 [252]), and other MPEG-4 audio formats such as TwinVQ [255] (Transform-domain Weighted INterleave Vector Quantization), WMA [256] (Windows Media Audio) or PAC [258] (Perceptual Audio Coder).

Another promising candidate, specifically tuned for voice applications with low bandwidth constraints, could be G.723.1 [236], which generally exhibits better quality and performance characteristics than GSM 06.10 [227]. G.723.1 [236], also called TrueSpeech 6.3/5.3, is a member in the TrueSpeech family of speech compression algorithms from DSP Group, Inc. [243], and produces digital voice compression levels of 20:1 and 24:1 respectively — with bandwidth requirements of 6.3 Kbps and 5.3 Kbps respectively, which is roughly half of what GSM 06.10 [227] requires.

It is adopted by the International Telecommunications Union, under the designation ITU-T G.732.1 [236], as part of the ITU-T H.324 [238] family of standards that address audio-visual communication in very low bit rate conditions, such as modem connections over public, traditional telephone systems — commonly referred to as POTS or Plain Old Telephone Systems. The same format is also used for the ITU-T H.323 standard [237] that addresses audio-visual communication over Local Area Networks as the low bit-rate speech technology recommendation, as discussed in [19]. If the licensing issues involved can be resolved, G.723.1 [236] would be an excellent alternative to GSM 06.10 [227].

Also, in the future, as the speech synthesis capabilities will evolve to become virtually indistinguishable from natural human speech, sound annotation transcripts might be used as the source for computer-synthesised speech as an alternative to recorded human speech. In this context, issues of intonation and emphasis are likely to present significant research challenges.

However, improvements in the AudioGraph presentations are not likely to stop at the technologies used to capture and distribute images and sound.

7.5 Other Improvement Opportunities

From a technical perspective, the management of large multimedia files sets some interesting challenges to explore. AudioGraph lecture files tend to become large — on the order or a few megabytes — as they typically incorporate a substantial amount of voice and image data. Saving such large files becomes time consuming, and may prompt us to consider alternatives to reduce this burden.

For instance, HyperCard [76] chooses to have its *stacks* and *cards* continuously synchronised with the disk as soon as each change is made. As a result, no *Save* command is necessary. Since all additions are appended to the stack file and all deleted objects are simply marked in place as deleted, HyperCard stacks benefit from being *compacted* every now and then.

This kind of approach would certainly reduce the amount of time required to save a large multimedia file, by having most of the disk accesses happen while the author is creating the lecture, potentially with a small penalty incurred in editing performance.

A potentially better alternative would be to perform an incremental save when the *Save* command gets invoked, by saving only the latest changes, rather than either saving each annotation to disk as it is edited or saving the whole lecture file even if only a very small portion of it has actually been changed. In such a scenario, slightly more disk space might need to be used, as deleted objects would have to be marked in-place rather than overwritten.

To mitigate this weakness, a mechanism for on-the-fly de-fragmentation could be used. For instance, a separate thread could be set up to monitor user activity. In periods of calm, when the user is not actively using the computer, the currently open lectures could be re-arranged on disk in order to use the minimum amount of space and eliminate any unused file sections. Similar techniques are used by commercial disk de-fragmentation utilities, such as Diskeeper [274] from Executive Software International.

On a Macintosh, files may contain two types of data; resources, stored in the *Resource Fork*, and plain data stored in the *Data Fork*. The Macintosh provides an OS-level API called the *Resource Manager* that provides facilities for dealing efficiently with resources.

To capture sound and images and manipulate them easily, the Macintosh version of the AudioGraph recorder makes use of resources. In the interest of cross-platform compatibility between the Macintosh and Windows AudioGraph recorders, future versions of the recorders may wish to support data-only lecture files as well, since the Windows platform does not support resource components in files, as the Macintosh does.

Another viable alternative might be to depart from the file-based document-storage model and adopt a database engine capable of efficiently handling multimedia data for storing all annotations. However, most relational database engines available currently are tuned for handling primarily small-footprint data, such as various numeric types and small string types, rather than BLOB's — Binary Large OBjects — such as sound clips, images and other multimedia data. Effectively, this approach would out-source the concerns related to efficient handling of potentially large media objects to the database engine and its associated compacting and maintenance tools.

It is quite likely that such an approach could yield significantly better loading and saving performance compared to the current solution. In addition, if educational institutions would make a commitment to use the AudioGraph system then multiple lecturers could store their course annotations in the same central database, relieving the burden on their personal workstations or network file servers. However, there are significant costs associated with the maintenance and management of databases that would need to be taken into account. A mixed approach could be envisaged as well, with lecturers having the option of editing lectures on their notebook computers without requiring a high-speed network connection and then being able to upload their new educational material to the central database once fast network services are available.

7.6 Summary

In this chapter, we have shown that there are many exciting avenues open for future AudioGraph-related research efforts.

We have suggested various evaluation and incremental improvement opportunities that, from a conceptual perspective, might be pursued in order to provide a sound foundation for refining the AudioGraph system. We argued that approaches for smooth integration in Learning Management Systems should be investigated, in order to best blend AudioGraphic content within the wider educational context.

We then focused on diverse media representation challenges, such as extensions to provide support for the Synchronised Multimedia Integration Language (SMIL 1.0 [187] and SMIL 2.0

[188]) and other alternative distribution formats, such as Macromedia Flash [88], Apple QuickTime [228] or MPEG-4 [249].

We have suggested that extended media support might also be investigated in order to benefit from alternative formats and advances in image, sound and video compression. We concluded this chapter with a number of other technical improvement suggestions regarding multimedia storage issues.

Appendix A. The AudioGraph System in Action

In this appendix, we provide an illustrated, systematic guide to using the AudioGraph system, from creating an AudioGraph presentation, through its editing and export, to playing it back in a web browser, using the system developed on the Apple Macintosh platform.

Although AudioGraph lectures may also be created from scratch, for greater impact we will follow the process of transforming a Microsoft PowerPoint [64] presentation into an AudioGraph lecture. As seen in Figure A.1, to achieve this, we must first export the presentation as a Macintosh Scrapbook file, which will contain the slides as a collection of images.

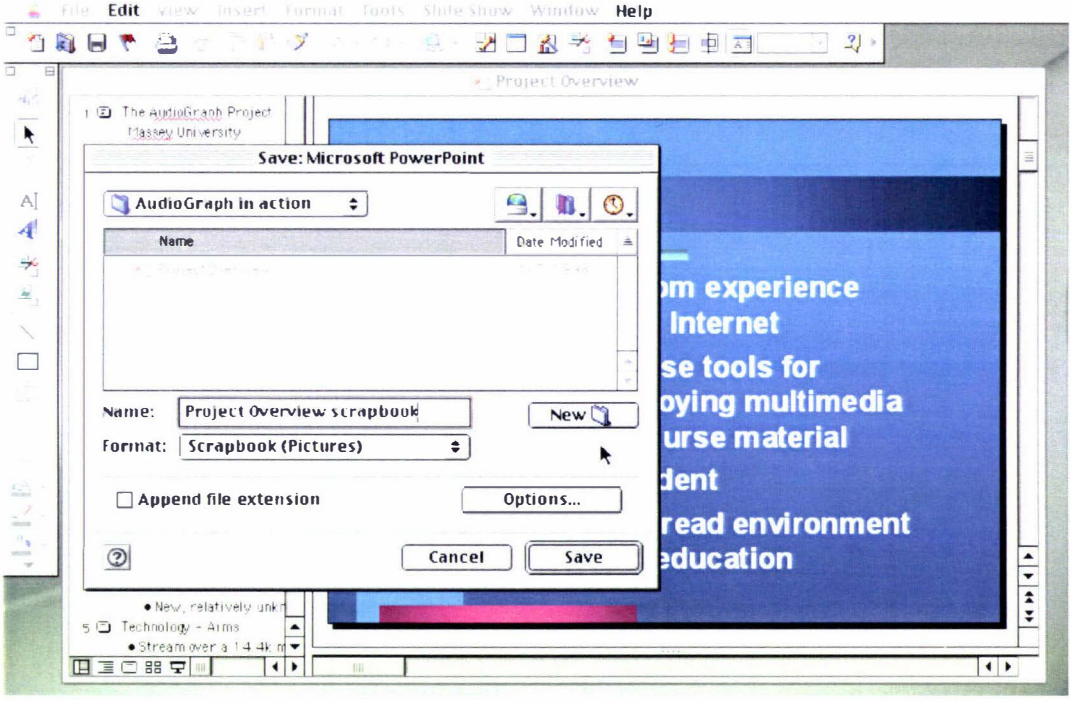


Figure A.1. Exporting a Microsoft PowerPoint presentation as a Macintosh Scrapbook file

Macintosh Scrapbook files are a convenient method of aggregating multiple pictures into the same file. The alternative — providing a series of image files in a folder — would have required the standard file open dialog to be modified to allow folder selections rather than file selections only.

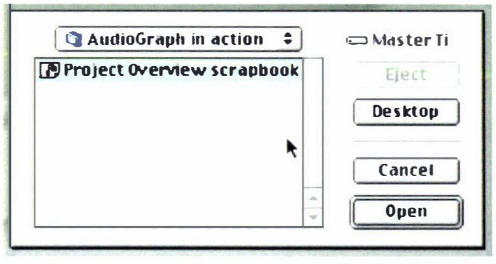


Figure A.2. Opening a Scrapbook file in the AudioGraph recorder

Once a PowerPoint [64] presentation is exported in this format, the AudioGraph recorder can open the resulting file. Alternatively, Scrapbook files containing any collection of pictures can be assembled using the standard Macintosh Scrapbook desk accessory.

When a Scrapbook file is opened in the AudioGraph recorder, as pictured in Figure A.2, each picture in the Scrapbook is associated with an episode, and it effectively becomes the background picture for that episode.

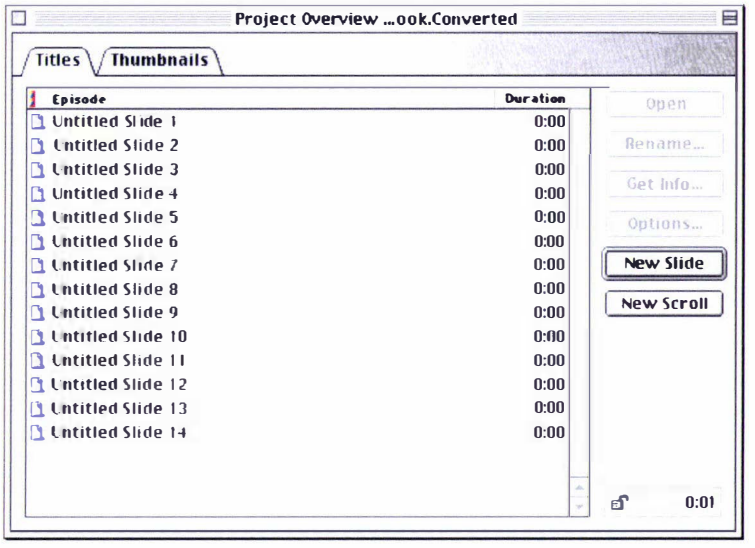


Figure A.3. A freshly imported Scrapbook file

In Figure A.3 you may notice that by default, the name of the lecture is the name of the scrapbook file with the string "`.Converted`" appended to it.

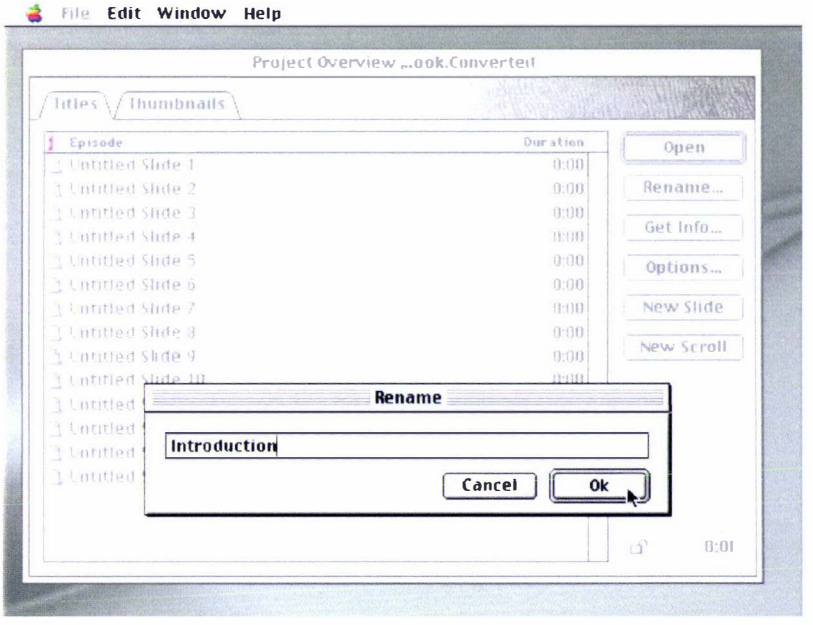


Figure A.4. Renaming episodes

Typically, at this stage, the episode titles are renamed to their appropriate names, as shown in Figure A.4. The thumbnails view, seen in Figure A.5, may be used make this task easier to accomplish by making visible at a glance what name should be most appropriate for a particular episode.

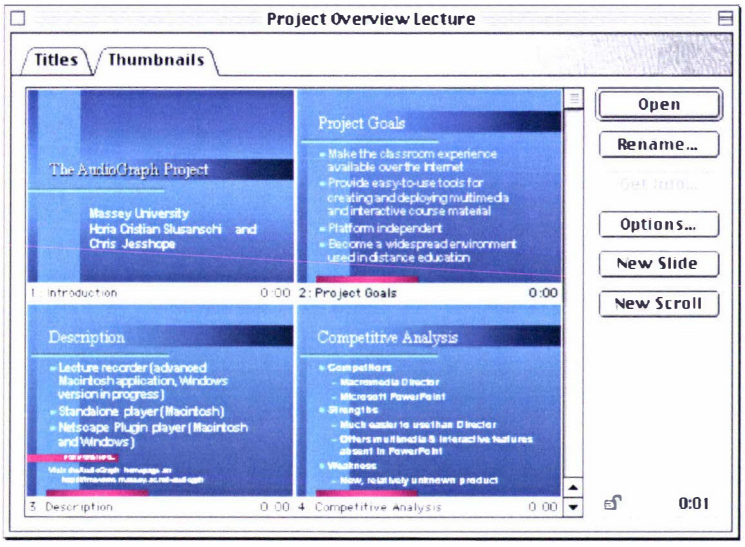


Figure A.5. Thumbnails view

At this stage, it is useful to save the lecture in the AudioGraph native format, and perhaps choose a more suitable name rather than the one provided by default in the conversion process.

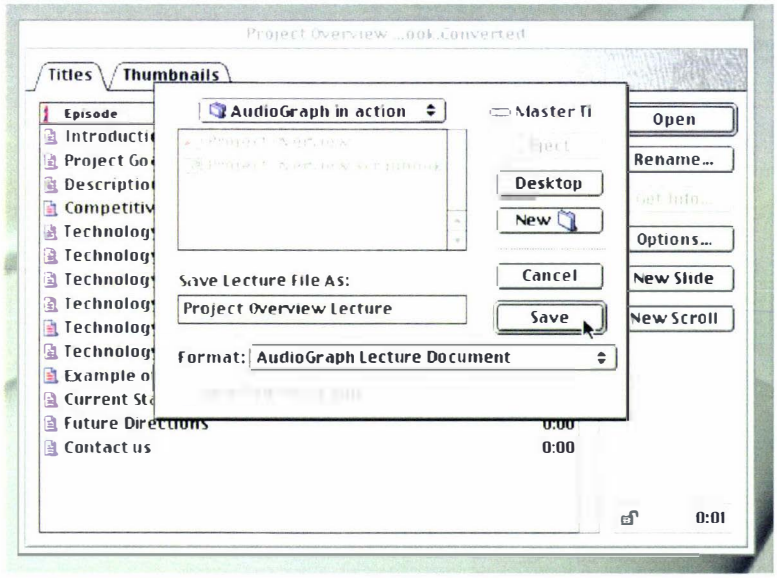


Figure A.6. Saving the lecture

Notice that at this stage, as illustrated in Figure A.6, the episode icons look slightly different, with a few red marks indicating that the episodes have been modified and not saved yet.

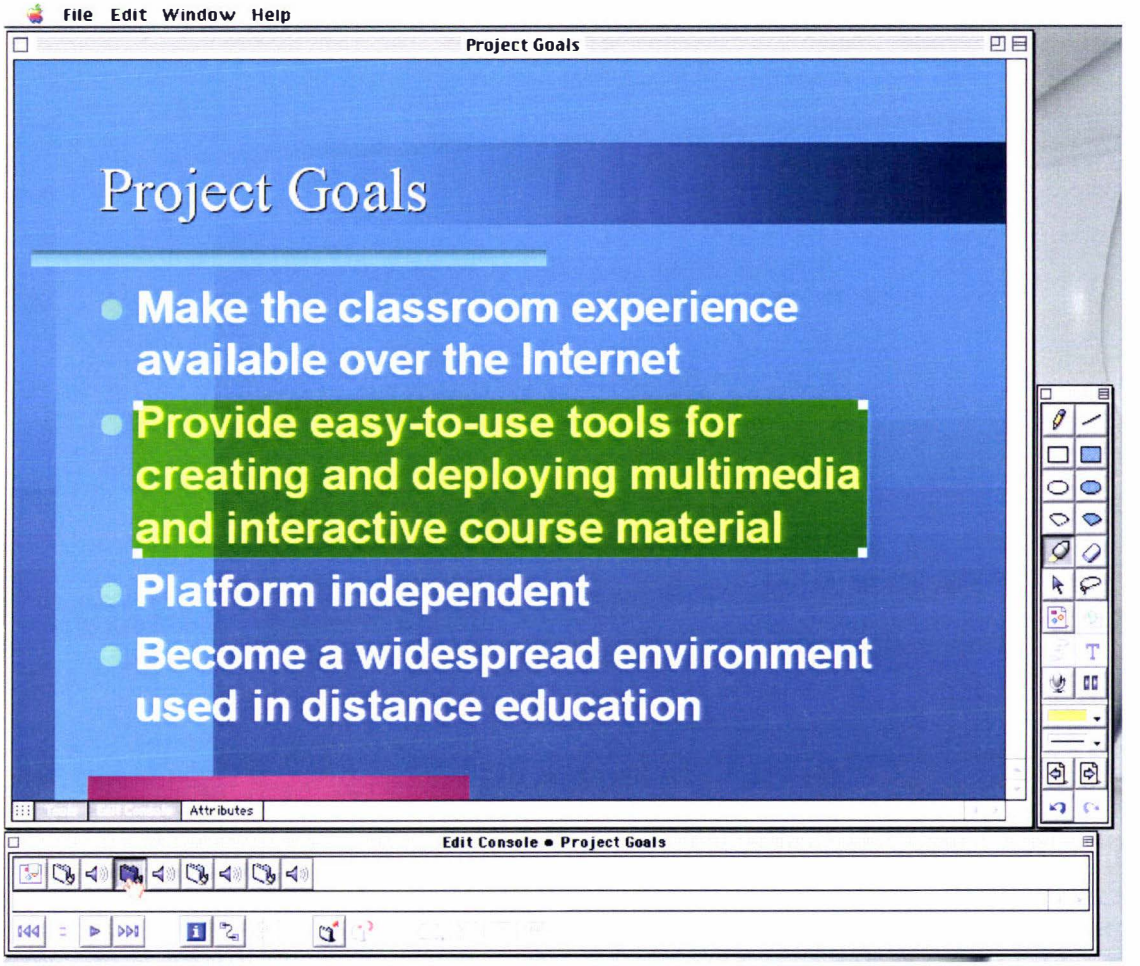


Figure A.7. Adding annotations

As soon as the lecture will be saved, the icons will lose the red markers, indicating that all the data is safely stored on disk, and no changes are pending. It is now appropriate to start adding annotations to the freshly created episodes, as seen in Figure A.7.

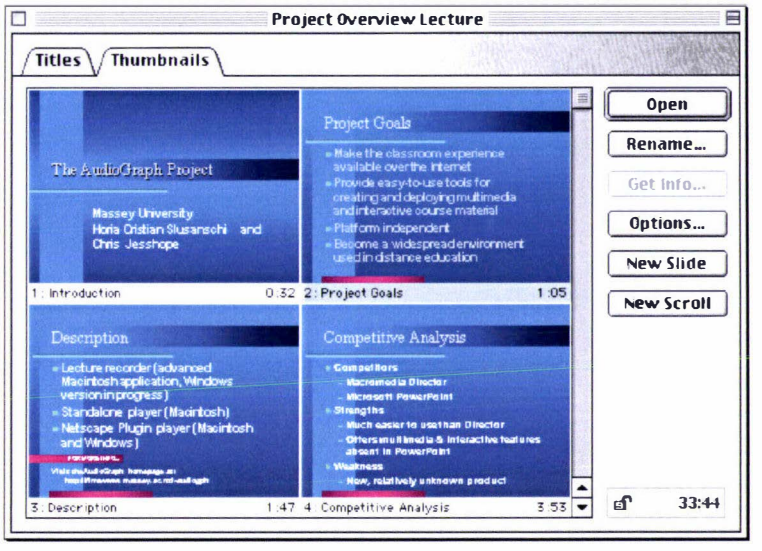


Figure A.8. Editing all episodes

Episode windows may be opened from the Lecture window either by clicking the *Open* button, or by double-clicking the desired episode title or thumbnail. You may notice that the individual episode durations are reflected in the lecture window, as well as the total lecture duration.

Once all episodes have been annotated to the author's satisfaction, as seen in Figure A.8, the lecture may be exported into the web-ready format. To accomplish this, simply choose the *Save As...* command from the File menu, and select the desired format for export, as shown in Figure A.9.

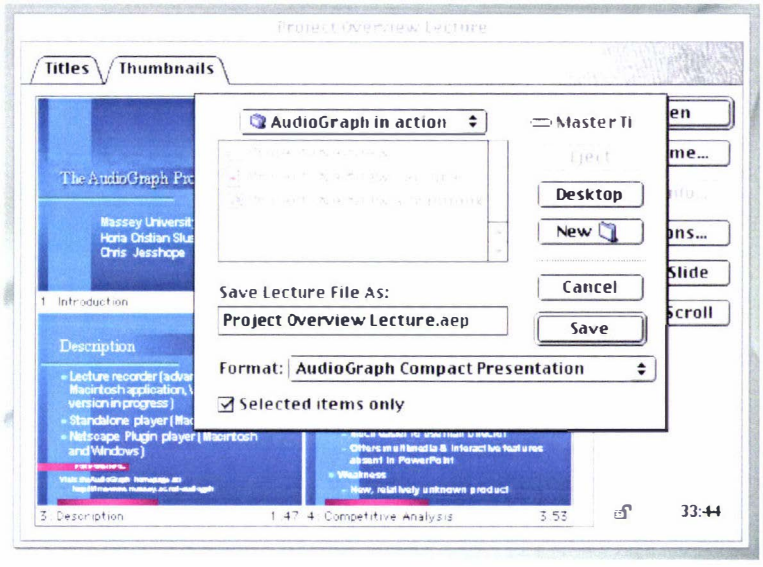


Figure A.9. Exporting in compact presentation format

AudioGraph lectures may be exported in *compact* or *expanded* format. Notice that an option is available to export only the currently selected items. This facility allows authors to release only subsets of their course material, as desired.

The *compact* format means that all episodes are presented in sequence in the same player instance.

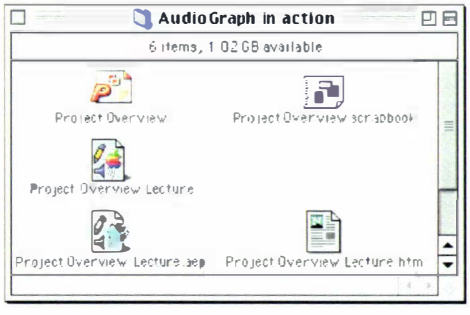


Figure A.10. The files generated in the compact export format

The recorder produces two files, pictured in Figure A.10, when exporting lectures in this format; one with the extension '.aep', which contains the AudioGraph presentation in web-ready format,

and the other with the '.htm' extension, which contains the appropriate HTML code to invoke the AudioGraph plug-in.

Figure A.11. The AudioGraph compact presentation viewed in the plug-in

By opening the HTML file in the web browser of your choice, while making sure that the AudioGraph plug-in is installed appropriately, you may view the AudioGraph episode, as seen in Figure A.11.

The HTML file may be included in course websites, and made available to the intended audience. Alternatively, the code in the HTML file may be integrated into web-based course management systems such as WebCT [68], TopClass [70], etc.

As an alternative to this approach, AudioGraph lectures may be transformed into websites in their own right, by using the *expanded* export format, as shown in Figure A.12.

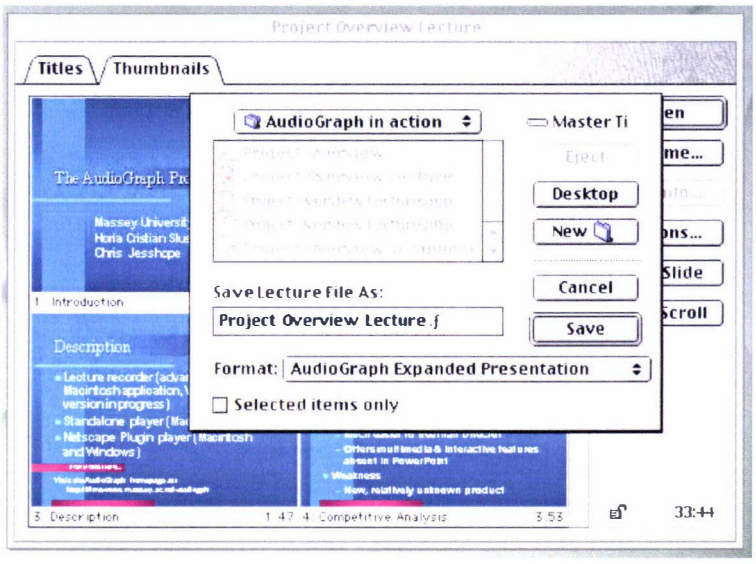


Figure A.12. Exporting a lecture in expanded format

This second export alternative, the *expanded* format, means that the lecture is transformed into a mini-website, with each episode saved in a separate file, as illustrated in Figure A.13. This approach has the benefit that if any individual episode must be re-released, it does not affect the rest of the lecture. In the *compact* export format, if an episode must be re-released, the whole presentation must be regenerated.

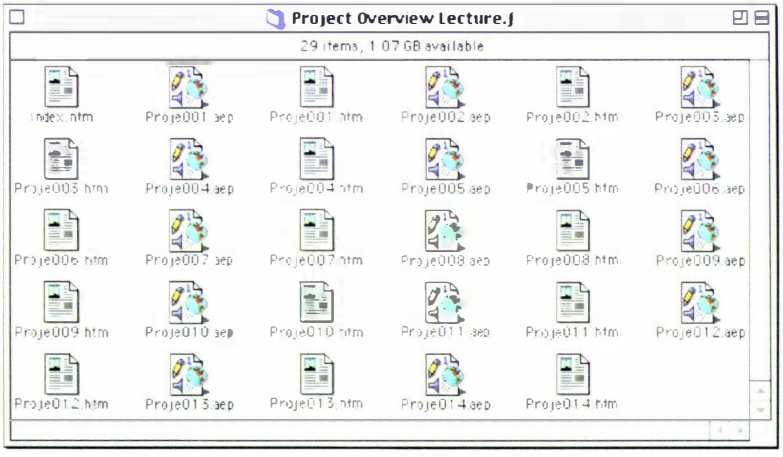


Figure A.13. The expanded format website

Notice that in this format, each episode generates two files just as for the *compact* export format, and an extra *Index.htm* file is created to aggregate the episodes into a coherent whole.

Students can take advantage of the *expanded* format to find particular episodes much easier by using the index page.

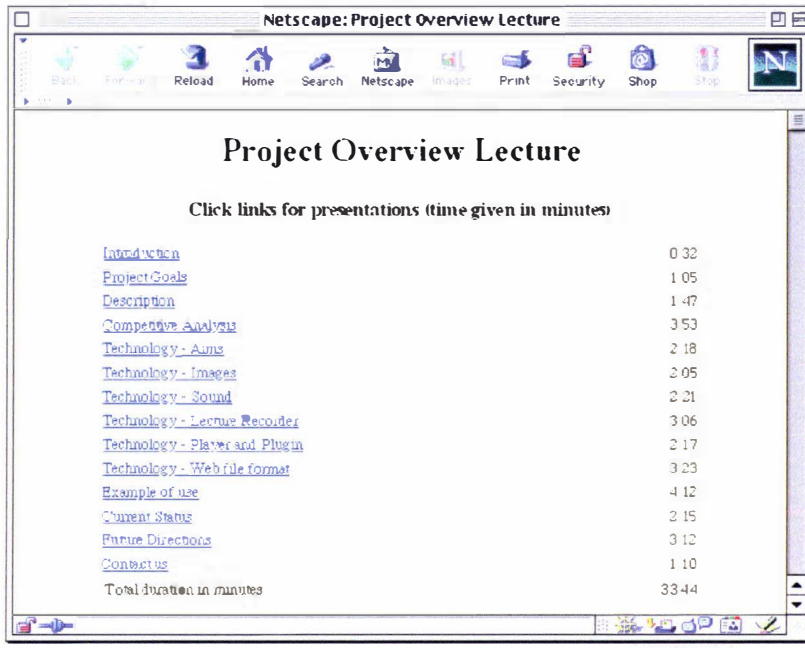


Figure A.14. The expanded lecture index

The index file, shown in Figure A.14, provides an overview of the course material and the individual and total presentation durations. By clicking the links on the left, the individual episode presentations are displayed in the browser window, just as seen in Figure A.11. Three extra buttons are provided in the HTML [191] pages in an *expanded* lecture website, to facilitate access to the next and previous episodes and back to the index.

Appendix B. QuickTime Technology

In the following sections, we will briefly review the underlying concepts of the QuickTime technology [228].

B.1 Atoms

QuickTime [229] atoms are basic containers that QuickTime uses to construct hierarchical data structures. A newly created QuickTime atom is like the root of a tree. Each subsequent QuickTime atom is contained in it and contains either data or other atoms. If a QuickTime atom contains other atoms, it is called a parent atom and the atoms it contains are called its child atoms. If a QuickTime atom contains data, it is called a leaf atom. Applications and other kinds of software can also use the QuickTime atom architecture to store data. QuickTime atoms are an effective implementation of the *composite* design pattern [127] (pp. 163).

Still image, movie frame and sound data, vector graphics and sprites are all examples of QuickTime atoms.

Sprites are an animated graphic created with QuickTime. With traditional video animation, we describe a movie frame by specifying the colour of each pixel. With sprite animation, we describe frames by specifying the images that appear at various locations. This object-based approach has also been embraced by the MPEG-4 specification [249].

Each sprite has properties that describe its location and appearance at a given time. During an animation sequence, the application modifies the sprite's properties to cause it to change its appearance and move around the screen. Changing sprite properties requires significantly less bandwidth than changing frame data would. Sprites may be mixed with still-image graphics to produce a wide variety of effects while using relatively little memory.

B.2 Media Structures

Traditional video consists of a continuous stream of data. A QuickTime [229] movie can be similarly constructed, but it need not be — a QuickTime movie may consist of streams of data taken from different sources, such as analogue video, still images stored on CD-ROM, and MIDI music. The movie is not the medium; it is the organizing principle.

QuickTime movies usually consist of several tracks. A track does not contain movie data; it contains only references to data that are stored elsewhere in leaf atoms as images, sounds, or any of the other supported media types. These data references constitute the track's media structure. Each track contains a single media structure and an edit list that arranges the media structure into a time sequence. Media structures are implemented as QT atom composites.

B.3 Components

QuickTime [229] supports plug-in components that free applications from needing to know about all the technologies and devices that QuickTime works with. Many QuickTime services, such as image compression and decompression, are provided by components. A component contains code, which can be made available globally to the whole system or locally to a particular application.

Each QuickTime component implements a defined set of features and presents specific interfaces to the technology it supports and to its client applications. Applications are thereby isolated from the details of implementing and managing various technologies.

QuickTime [229] is supplied with around 200 built-in components. It also contains an API that lets developers create new components if they wish. For example, users could create components to support particular data encryption algorithms. Multiple applications could then perform encryption by connecting to those components instead of having to implement the algorithms themselves.

Applications gain access to components by calling the system-level Component Manager [156]. The Component Manager [156] lets users define and register types of components and communicate with components using a standard interface. Once an application is connected to a component, it calls the component directly. When developers create a new component class, they define a standard function-level interface for that type. This means that a variety of applications can access the component with full confidence that it will work as designed.

B.4 Time Management

Time management is an important and sometimes complex part of the implementation of QuickTime [229] movies. For a substantial proportion of movies, the correct play rate is the rate at which human actions appear natural and objects fall to the ground with normal acceleration, as observed by Sir Isaac Newton.

However, what should be the correct play rate of a movie that shows spreadsheet data charted over time or a map of the earth that recapitulates continental drift? This problem is deepened by the differing clock speeds of various platforms and the need to decompress data in real time, all of which affect time scales.

To manage the time dimension of movies, QuickTime defines time coordinate systems, which anchor movies and their media data structures to a common temporal reality, the second. A time coordinate system contains a time scale that provides the translation between real time and the apparent time in a movie. Time scales are marked in time units. The number of units that pass per second quantifies the scale — that is, a time scale of 42 means that 42 units pass per second

and each time unit is 1/42 of a second. Time coordinate systems also specify durations, which are the movie or media structure lengths expressed in terms of the number of time units contained.

Any particular point in a movie can then be identified by the number of time units elapsed to that point. Each track in a movie contains a time offset and a duration, which determine when the track begins playing and for how long.

Each media structure has its own time scale, which determines its number of samples per second. The Movie Toolbox maps each type of media data from the movie's time coordinate system to the media structure's time coordinate system.

Appendix C. The Emergence of XML and its Related Technologies

In recent times, a number of technological advances have converged to make possible new kinds of interaction between humans and have deeply affected our means of communication. Until the invention of electrical and then electronic means of communication, information was propagated between humans at a relaxed pace. It took days, or even months for messages between people to reach their destinations. With the advent of the Internet, e-mail and the Word-Wide-Web, information is flowing between us in a matter of minutes, seconds or even fractions of seconds. In effect, our raw information sharing ability has reached the fundamental limitation of electromagnetic wave propagation — the speed of light. Some of our current technological efforts are devoted to increasing the bandwidth of our communication channels, so that we may exchange ever more information in the shortest possible time.

Although the ability to transfer information from one location to another is vital, no less important is the capacity to create and express it. In one of its simplest forms, electronic communication only requires the transmission of binary signals across the communication medium. Effectively, current computers primarily deal with binary numbers. However, messages expressed as mere sequences of binary values are very cumbersome to decode and interpret for humans, as they do not typically represent symbols with which we are familiar.

Therefore, in order to make electronic communication meaningful, various encoding schemes have been developed to match alphabetic or ideographic symbols with numerical representations. Using this approach, streams of binary values can be interpreted as text messages. Due to its simplicity, the earliest forms of communication over the Internet, such as e-mail, have made use of text messages.

In an effort to standardise the myriad encoding schemes that have been developed in the years before the Internet started bringing computers together, the Unicode [164] effort seeks to provide a universally recognised encoding standard for virtually every alphabet and script currently known.

However, mere text messages are not enough to convey the full richness of human expression, and therefore markup languages, such as SGML [166] and HTML [190] have been developed in an attempt to annotate text with semantic markers in a standardised fashion. For instance, using such a mechanism, we could specify sections of text that should be emphasised or others that should be organised in tabular form.

Although HTML [190] is the ubiquitous language in which virtually all the documents that make up the World-Wide-Web are expressed, in its historical form, it suffers from a few

drawbacks that make it less than ideal for a more integrated cooperation between humans and computers.

Since in the earliest days of the World-Wide-Web there were no specialised authoring tools that would produce HTML [190] documents, many authors have written large numbers of web pages using plain text editors. As a result, in order to cope with the inevitable human errors, web browsers have adapted to support the myriad non-standard idioms and imperfect syntactical constructs that large numbers of web pages might still be using.

Given the steadily increasing performance of computers, it becomes feasible and quite likely desirable as well to develop new means of interaction with software applications, and bring about a more collaborative relationship between humans and software. In their words, Tim Berners-Lee, James Hendler, and Ora Lassila [171] refer to this vision as the *Semantic Web*: "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

As could well be expected, a very relaxed syntax makes it very difficult to devise software that would be able to interpret the meaning of messages on its own. Unless precise syntax would be used, with clearly defined semantic associations, it would remain extremely difficult to devise software that unambiguously discriminates the meaning of arbitrary messages or documents.

C.1 Extensible Markup Language (XML)

In order to address such concerns, the eXtended Markup Language (XML) [167] effort has been undertaken. In the following paragraphs, we will review the main concepts conveyed in the short introduction to XML, *XML in 10 points* [168] provided by the W3C [163], which gives a good overview of the intent of XML.

The most important guideline when dealing with XML is that it should be used for structuring data. Text documents, spreadsheets, address books, configuration parameters, financial transactions, and technical drawings are all forms of structured data. XML consists of a set of rules — guidelines or conventions — for designing data structures expressed in textual form. In itself, XML is not a programming language, and anyone can use or learn it regardless of programming skills. Using XML, software components may read, transform and generate data, with the certainty that it is structurally unambiguous. Since there has been a vast amount of effort spent in developing XML, it is specified to avoid common pitfalls in language design: it is extensible, platform-independent, and it supports internationalisation and localization. XML is fully Unicode-compliant [164].

Taking advantage of the popularity of HTML [190], XML uses nearly identical syntax, which instantly raises into the millions the number of people able to interpret XML documents at a

basic level without any special training. Just as HTML does, XML uses tags (words bracketed by '<' and '>') and attributes (of the form name="value") as its primary syntactical constructs. However, while in HTML the tag and attribute semantics are well defined [191], down to the recommended rendering behaviour for browsers, XML uses the tags and attributes only to delimit and structure data items, and passes the responsibility for semantic interpretation of the data to the applications that manipulate it. In other words, system designers are free to define the precise semantics of the elements and attributes in their XML documents as best suits their needs.

Although XML documents and messages are represented as text, in normal use humans should not be required to read them. Programs that produce spreadsheets, address books, and other structured data often store that data on disk, using either a binary or text format. One advantage of a text format is that it allows people, should the need arise, to look at the data without having to use the program or the platform in which it was created; if need be, text can be examined and edited with any plain text editor. Using this approach, the task of debugging data persistence issues for applications that use XML documents is greatly simplified for developers. They would no longer require special tools for examining files in binary format, and since XML is standardised, they would not need to familiarise themselves with any proprietary data-structuring text format either. Just like HTML, XML files are text files that people should not have to read, but may do so when the need arises. In contrast with HTML, the syntax rules for XML documents are strict. For instance, a forgotten closing tag, or an attribute without quotes renders an XML document syntactically invalid, while in HTML such practice is tolerated and it is even explicitly allowed in frequent instances [191]. The official XML specification [169] forbids applications from trying to second-guess the creator of an invalid XML document; if the file is not syntactically correct, an application must abandon any further processing and report an error.

Unlike custom binary data formats, XML is verbose by design. Since XML is a text-based data-structuring format and it uses tags to delimit the data, XML documents are consistently larger than comparable binary representations. That was a conscious decision made by the designers of XML. The advantages of a text format are evident, as already outlined, and the disadvantages can usually be compensated for in different ways. Given the steady advance in storage and microprocessor technology, vastly larger amounts of storage are cheaply available, and freely available compression utilities can be used to compress and decompress files very well and very fast. Given the textual nature of XML documents, they are typically excellent candidates for data compression given the large information redundancy in such messages. In addition, communication protocols such as modem protocols (V90 [213], and other series V ITU

recommendations [214]) and HTTP/1.1 [184], the core protocol of the Web, can compress data on the fly, saving bandwidth just as effectively as a binary format.

To gain the full benefit of XML standardization, a family of technologies has been defined around it. XML 1.0 [169] is the specification that defines what *tags* and *attributes* are and how they are structured to form valid documents and elements. Beyond XML 1.0 [169], the XML family is a growing set of modules that offer useful services to accomplish important and frequently used tasks. XLink [174] and XPath [175] describe standardised advanced ways to add hyperlinks to XML documents. XPointer [176] is a syntax specification in development for pointing to parts of an XML document. An XPointer is a bit like a URL [196], but instead of pointing to documents on the Web, it can point to pieces of data inside an XML document. CSS [194], the style sheet language, is applicable to XML as it is to HTML. XSL [177] is the advanced language for expressing style sheets. It is based on XSLT [178], a transformation language used for rearranging, adding and deleting tags and attributes. The DOM [179] is a standard set of function calls for manipulating XML [169] (and also XHTML 1.0 [192] and XHTML 1.1 [193]) files from a programming language. XML Schemas [180] help developers define the structures of their own XML-based data structures, and even more modules and tools are available or under development.

The development of XML [169] started in 1996 and has been a W3C [163] Recommendation since February 1998, which might lead to suspicions that this is a rather immature technology. In fact, the technology is not quite so new. Before XML [169] there was SGML [166], developed in the early '80s, an ISO standard since 1986 and widely used for large documentation projects, most notably in the aerospace and defence industries. The development of HTML [191] started in 1990. The designers of XML simply took the best parts of SGML, guided by the experience with HTML, and produced something that is no less powerful than SGML [166], and vastly more regular and simpler to use. In addition, although SGML [166] is mostly used for technical documentation and much less for other kinds of data, XML [169] appears to be used in precisely the opposite fashion for the time being, as SGML-based systems have not yet been migrated to use XML.

There is an important XML application that is a document format: W3C's [163] XHTML 1.0 [192] and XHTML 1.1 [193], the successor to HTML [191]. XHTML has many of the same elements as HTML. The syntax has been changed slightly to conform strictly to the rules of XML, and the semantics of the XHTML tags are largely inherited from HTML [191].

XML [169] has a modular nature, facilitating the definition of new document structures by combining and reusing other structures. Since two document structures developed independently may have elements or attributes with the same name, care must be taken when

combining such formats. To disambiguate names when combining document structures, XML provides a *namespace* mechanism, just as other programming languages do, such as C++. XSL [177] and RDF [173] are good examples of XML applications that make full use of namespaces. XML Schema [180] is designed to mirror this support for modularity at the level of defining XML [169] document structures, by making it easy to combine two schemas to produce a third, which covers a merged document structure.

XML [169] is the foundation for the Resource Description Framework (RDF [173]) and the Semantic Web [172]. W3C's [163] RDF [173] is an XML application that supports resource description and metadata applications, such as music play-lists, photo collections, bibliographies and so on. For instance, relying on RDF [173], a user could identify people in Web-based photo albums using information from a personal contact list; and a mailing software agent could subsequently use that information to send messages automatically to the people involved, indicating where their photos are available on the Web. Just as HTML [191] integrated documents, menu systems, and forms applications to launch the original Web, RDF integrates applications and agents into one Semantic Web [172]. Just as humans need to agree on the meanings of the words they use in their communications, software agents need mechanisms for agreeing on the meanings of terms in order to communicate effectively. Formal descriptions of terms in a certain subject area — shopping or manufacturing, for example — are called *ontologies* and are a necessary part of the Semantic Web. RDF [173], *ontologies*, and the representation of meaning so that humans and computers can interact better are all topics of the Semantic Web Activity [172].

A significant benefit of XML [169] and its associated technologies is that they are license-free, platform-independent and well supported by industry. Given this widespread support, when choosing to use XML [169] and its associated technologies in new projects, organizations have access to a large and growing community of tools — some of which may already do precisely what would be required for the task at hand — and engineers experienced in the technology. Opting for XML [169] is similar to choosing SQL [138] for databases: the user would still have to build her own database, the associated programs and procedures that manipulate it, and there are many tools available and many people who can help in the software development and maintenance processes. Since XML [169] is license-free, software can be built around it without having to pay any licensing fees whatsoever. The large and growing industry support also means that organizations are not necessarily tied to a single vendor, which is of vital importance to enterprises of all sizes. Although XML [169] may not always be part of the best solution from a raw processing performance perspective, it is certainly worth considering for most future software projects due to its flexibility.

An interesting parallel can be drawn form the evolution of semiconductor technology. As the levels of performance have dramatically increased through constantly shrinking feature sizes and ever lower voltages, microprocessors have evolved from components containing intellectual property from a single supplier toward highly customised chips that integrate multiple intellectual property blocks potentially coming from different vendors, possibly even using different process technologies on the same die, creating custom systems-on-a-chip [270]. In other words, the fundamental values of competition have shifted from a singular focus on performance to customisation, reliability and convenience.

Personal computers have moved through a similar kind of evolution. In the not so distant past, as performance used to be of utmost importance, vertical suppliers had the advantage, since they could design, build, customise and tune all the components of their products for optimum performance.

However, as the raw performance offered by technological advances began to soar, the importance of vertical integration diminished, with the advantage shifting to the organizations that could best integrate cheaper and faster standard components from various suppliers, as their products began to be able to perform similar tasks with similar overall performance, even if the integration did not lead to the utmost performance being squeezed out of the hardware.

The key lies in the standardization of various interfaces so that components from multiple vendors can be integrated seamlessly, even though the use of such interfaces may incur performance penalties. Effectively, more value is placed upon ease of integration than on sheer performance.

We can see the same dynamic at play with the evolution of web-based education systems and their associated data representation formats, of which the AudioGraph is one example that places emphasis on sheer performance and bandwidth economy, and the emergence of standards such as SMIL 1.0 [187], SMIL 2.0 [188] and SVG [185], which promise simplified integration from arbitrary heterogeneous data sources.

This recurring pattern of the increased importance of the ease of customisation — the ability to rapidly adapt to change — over sheer performance or efficiency as the underlying raw technological performance rises can also be seen at work in the evolution of XML and its related technologies.

In the past, due to constraints in the amount of storage space and network bandwidth capacity available, it would have been unthinkable to adopt such a 'wasteful' data storage format as XML, since it is text-based, verbose and in consequence it generally uses considerably more storage space than equivalent binary data structures. Proprietary file formats were in widespread use, as they promised higher efficiency and better use of the scarce available resources. Many

proprietary file formats are still enjoying tremendous popularity, such as PDF [199] and GIF [225], although use of the latter is declining due to licensing issues and the increased availability of better alternatives such as the license-free PNG [218].

However, as the volume of communication in electronic form between computers and organizations began to increase and the capacity of the data links rose, the most important values started to shift from utmost data storage and transmission efficiency to standardization, flexibility and ease of processing.

The importance of license-free standards cannot be overstated. The key to the success of XML technology lies in the standardization of the expression of data structures, which can also be seen as the software interfaces between information processing systems.

Gordon Moore, co-founder of Intel [269], noted in 1965 that the number of transistors per square inch on integrated circuits had doubled every year since the integrated circuit was invented. At the time, Moore predicted that this trend would continue for the foreseeable future. In subsequent years, although the pace of development relented somewhat data density has consistently continued to double approximately every 18 months. This observation is currently known as *Moore's Law* [268], which most experts, including Moore himself, expect to hold for at least another two decades. This exponential growth in technological capability, unprecedented in the history of humanity, has led to this dynamic of the value of ease of customisation and flexibility rising over that of mere performance for the sake of ultimately greater capability.

Just as the standardization of the hardware interfaces between the various components of computers has brought vast advances in their flexibility and the speed with which they can be customised for new purposes, the standardization of interfaces between software agents is promising to bring similar kinds of improvements in the realm of the interaction between humans and computers.

Since education is one of the most important applications of communication, it is only natural to expect that the evolution of XML-related technologies would also bring advances for its benefit. SVG [185] and SMIL 1.0 [187] and 2.0 [188] are applications of XML that are likely to have a significant impact on the future of education. Given the pervasiveness of computers and the Internet, it is likely that the boundaries between traditional instructor-based education and web-based education will continue to blur and reshape the concept of education itself.

C.2 Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG [185]) is an application of XML [169] for describing two-dimensional graphics. SVG allows for three types [186] of graphic objects:

- Vector graphic shapes (e.g., paths consisting of straight lines and curves). Graphical objects can be grouped, styled, transformed and combined with previously rendered objects.
- Images.
- Text, which can contain constructs expressed in any XML namespace [170]. This approach enhances the accessibility and capability to search for SVG graphics.

The full feature set of SVG [186] includes nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility capabilities.

SVG drawings can be dynamic and interactive. The Document Object Model (DOM) for SVG, which includes the full XML DOM [179], allows for straightforward and efficient vector graphics animation via scripting. A rich set of event handlers such as *onmouseover* and *onclick* can be assigned to any SVG graphical object, giving very good interactivity capabilities. Because of its compatibility and leveraging of other Web standards, features like scripting can be done on SVG elements and other XML elements [169] from different namespaces [170] simultaneously within the same Web page.

Bibliography

AudioGraph-related Resources

[1] Jesshope, C. R., Shafarenko, A., (1997). *Web Based Teaching: a minimalist approach*, Proc. Second Australasian Conference on Computer Science Education, Association for Computing Machinery, ISBN: 0-89791-958-0, pp. 16-23.

[2] Segal, J., (1997). *An evaluation of a teaching package constructed using the Audiograph. a web-based lecture recorder.* Assoc. for Learning Technology Journal, Department of Computing, University of Surrey, UK, pp. 32-42. Also available online: http://www.nzedsoft.com/Papers/Segal.pdf. 20 January 2004.

[3] NZEdSoft's Home Page. Online. New Zealand Educational Software. Available: http://www.nzedsoft.com/. 20 January 2004.

[4] More Demonstrations. Online. New Zealand Educational Software. Available: http://www.nzedsoft.com/demonstrations.html. 20 January 2004.

[5] Pearson, M., Jesshope, C. R., (1998). *Multi-campus teaching using computer networks*, Proc. of the Third Australasian Conference on Computer Science Education, Association for Computing Machinery, July 1998, pp. 106 - 111.

[6] Jesshope, C. R., Slusanschi, H. C., (1998). *Recording Formal Lectures for Publication on the Web*, Proc. of the DEANZ 1998 Conference, Rotorua, July 1998, pp. 167-179.

[7] Jesshope, C. R., Shafarenko, A., Slusanschi, H. C., (1998). *Low-bandwidth multimedia tools for web-based lecture publishing*, IEE Engineering Science and Educational Journal, 7 (4), pp. 148-154.

[8] Jesshope, C. R., Shafarenko, A., Slusanschi, H. C., (1998). *Low-bandwidth multimedia tools for web-based lecture publishing*, IEE Computing and Control Engineering Journal, 9 (4), pp. 156-162. Also available online: http://www.nzedsoft.com/iee%20paper/frame2.htm. 20 January 2004.

[9] Jesshope, C. R., Slusanschi, H. C., (1998). *Distance Education Using the AudioGraph Authoring Tool*, Proc. Intl. Conf. on Multi-Media Information Systems in Practice, ISBN 981-4021-53-9, Springer, Hong Kong, December 1998, pp. 266-278.

[10] Jesshope, C. R., (1999). *Web-based Teaching - Tools and Experience*, Australian Computer Science Communications, 21, (1), Proc. Australasian Computer Science Conference, ACSC99, ISBN 981-4021-54-7, Springer, Auckland, January 1999, pp. 27-38.

[11] Jesshope, C. R., Gehne, R., (1999). *Recording Lectures for Serving on the Web*, Proc. RUFIS 1999, (CD published by Northern Arizona University) October 21-24 1999, Flagstaff, USA.

[12] Jesshope, C. R., (2000). *TILE -Technology Integrated Learning Environments*, (Invited Paper), Proc. 7th IDEA Workshop, University of Adelaide, February 4-7 2000, Adelaide, Australia, pp. 39.

[13] Jesshope, C. R., (2000). *The use of streaming multi-media in microelectronic education*, Microelectronics Education, Kluwer Academic, London, ISBN 0 7923 6456 2, pp. 45-48.

[14] Jesshope, C. R., (2000). *The use of multi-media in internal and extramural teaching*, Proc. Lifelong Learning Conference, Central University of Queensland, Brisbane, Australia, ISBN 187 6674 06 7, pp. 257-262.

[15] Gehne, R., Jesshope, C. R., (2000). *Tools for the production of small-footprint, low-bandwidth, streaming multi-media for distance education*, Proc. Lifelong Learning Conference, Central University of Queensland, Brisbane, Australia, ISBN 187-6674-06-7, pp. 240-244.

[16] Jesshope, C. R., Heinrich, E. and Kinshuk, (2000). *On-line Education using Technology Integrated Learning Environments*, Proc. DEANZ 2000 Conference, Otago University, New Zealand, 27-29 April 2000. Also available online: http://www-tile.massey.ac.nz/publications/deanz_paper.pdf. 20 January 2004.

[17] Jesshope, C. R., (2000). *Integrated tools for on-line education*, Proc. IWALT 2000, ISBN 0-7695-0653-4, IEEE Computer Society (Los Alamitos CA, USA), pp. 205-8.

[18] Jesshope, C. R., (2000). *Using AudioGraph in On-line Teaching*, Proc. Open Learning Conference, Brisbane, Australia, pp. 315-320, Learning Network Queensland (Brisbane, Australia).

[19] Jesshope, C. R., Liu, Y. Q., (2001). *High Quality Video Delivery over Local Area Networks With Application to Teaching at a Distance*, Intl J. of Electrical Engineering Education (IJEEE), Vol 38(1), Manchester University Press, ISSN: 0020-7209, pp. 11-25.

[20] Heinrich, E., Jesshope, C.R. and Walker, N., (2001). *Teaching Cognitively Complex Concepts: Content Representation for AudioGraph Lectures*, Proc. ED-MEDIA 2001, pp. 714-719.

[21] Foundations of Computer Science 159.102. Online. Massey University. Available: http://www-ist.massey.ac.nz/~crjessho/foundations/index.html. 1 December 2003.

[22] Networks for Parallel Computers. Online. Massey University. Available: http://www-ist.massey.ac.nz/~crjessho/comp_arch/. 1 December 2003.

[23] Personal conversations with Paul Lyons on alternative means of using existing course material as a starting point for AudioGraph lectures, Massey University, 1998.

Education Resources

[24] Draper, S., (1997). *Prospects for summative evaluation of CAL in higher education.* ALT-J 5(1), pp. 33-39.

[25] Gunn, C., (1997). *CAL evaluation: future directions.* ALT-J 5(1), pp. 40--47.

[26] Maurer, H., (2002). *What Have we Learnt in 15 Years About Educational Multimedia?.* Keynote Paper, Proc. ED-MEDIA 2002, pp. 2-7.

[27] Learning Theories. Online. Emerging Technologies. Available: http://www.emtech.net/learning_theories.htm. 1 December 2003.

[28] Learning Theories. Online. Darren Forrester & Noel Jantzie. Available: http://www.ucalgary.ca/~gnjantzi/learning_theories.htm. 1 December 2003.

[29] Perspectives on Learning. Consequences on instructional design based on behaviorism, cognitivism, constructivism. A psychological approach. Online. Boukje Bruinsma and Wiert Berghuis. Available: http://www.geocities.com/learningenvironments/. 1 December 2003.

[30] Gardner, H., (1993). *Multiple Intelligences: The Theory in Practice.* New York: BasicBooks, a division of HarperCollins Publishers, ISBN: 0-465-01822-X.

[31] Kolb, D. A., (1984). *Experiential Learning: Experience as the Source of Learning and Development.* Prentice Hall, Englewood Cliffs, NJ, ISBN: 0-132-95261-0.

[32] Litzinger, M. E., Osif, B., (1993). *Accommodating diverse learning styles: Designing instruction for electronic information sources.* In *What is Good Instruction Now? Library Instruction for the 90s.* Shirato, L. (Ed.). Pierian Press, Ann Arbor, MI, ISBN: 0-87650-327-X.

[33] Learning Styles. Online. Jessica Blackmore. Available: http://cyg.net/~jblackmo/diglib/styl-a.html. 1 December 2003.

[34] Hartman, V. F., (1995). *Teaching and learning style preferences: Transitions through technology.* VCCA Journal 9, No. 2, summer, pp. 18-20.

[35] Dabbagh, N., (2001). *Web-Based Course Authoring Tools and Online Learning: A Technological and Pedagogical Perspective.* Invited Keynote speech at the The World Internet and Electronic Cities Conference, Kish Island, Iran. Online. Nada Dabbagh. Available: http://mason.gmu.edu/~ndabbagh/WIECC-keynote.ppt. 1 December 2003.

[36] Collis, B., (1997). *Pedagogical re-engineering: a pedagogical approach to course enrichment and redesign with the WWW*. Educational Technology Review, Autumn/Winter (8), pp. 11-15.

[37] Bruce, B. C., Levin, J. A., (1997). *Educational Technology: Media for Inquiry, Communication, Construction, and Expression*, Journal of Educational Computing Research, Vol. 17(1), pp. 79-102. Also available online: http://alexia.lis.uiuc.edu/~chip/pubs/taxonomy/. 1 December 2003.

[38] Dewey, J., (1943). *The child and the curriculum / The school and society*. Chicago: University of Chicago Press.

[39] LESTER: Taxonomy for Learning Sciences and Technology. Online. Learning Science and Technology Repository. Available: http://antioch.rice.edu/etrac/lester/taxonomy.html. 1 December 2003.

[40] etrac | educational technology research and assessment cooperative. Online. Rice University. Available: http://antioch.rice.edu/etrac/index.html. 1 December 2003

[41] Weippl, E., (2002). *The Transition from Computer-Based Training to eEducation*, Proc. ED-MEDIA 2002, pp. 2034-2039.

[42] Iiyoshi, T., Hannafin, M. J., (2002). *Cognitive Tools and User-Centered Learning Environments: Rethinking Tools, Functions, and Applications*, Proc. ED-MEDIA 2002, pp. 831-836.

[43] Parsley, M., (2002). *Education Communities, Government and Technology: Partnering for a Better Tomorrow*, Proc. ED-MEDIA 2002, pp. 1545-1547.

[44] Raphael, C., (2002). *"Citizen Jane": Rethinking Design Principles for Closing the Gender Gap in Computing*, Proc. ED-MEDIA 2002, pp. 1609-1614.

[45] Grasha, A. F., Richlin, L., (1996). *Teaching with Style*, Alliance Publishers, Pittsburgh, PA, ISBN: 0-964-50711-0.

[46] Boyle, T., (2003). *Design principles for authoring dynamic, reusable learning objects*. Australian Journal of Educational Technology, 19(1), pp. 46-58. Also available online: http://www.ascilite.org.au/ajet/ajet19/boyle.html. 30 November 2003.

[47] IEEE 1484.12.1-2002 (2002). *Draft Standard for Learning Object Metadata*. Online. Available: http://grouper.ieee.org/p1484/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf. 30 November 2003.

[48] Multimedia Authoring Systems FAQ. Online. J. Siglar. Previously available at: http://www.tiac.net/users/jasiglar/MMASFAQ.HTML. 21 Feb. 2002. Also available in archived form at: http://www.cs.uu.nl/wais/html/na-dir/multimedia/authoring-systems/.html. 1 October 2003.

[49] Smith, P. L., Ragan, T. J., (1999). *Instructional Design*. Wiley Text Books, Second edition, 1999, ISBN: 0-471-36570-X, pp. 2.

[50] West, C. K., Farmer, J. A., Wolff, P. M., (1991). *Instructional Design: Implications from Cognitive Science*. Prentice Hall, Englewood Cliffs, NJ, ISBN: 0-134-88578-3.

[51] Merrill, M.D., (2003). *First principles of instruction*. Submitted for publication to Educational Technology Research & Development (http://www.aect.org/Intranet/Publications/index.asp). Also available online: http://www.id2.usu.edu/Papers/5FirstPrinciples.PDF. 30 December 2003.

[52] Reigeluth, C.M., (1999). *Instructional Design Theories and Models: A New Paradigm of Instructional Theory*, Vol. II, Lawrence Erlbaum Associates, Mahwah, NJ, ISBN: 0-805-82859-1.

[53] van Merriënboer, J. J. G., (1997). *Training Complex Cognitive Skills*. Educational Technology Publications, Englewood Cliffs, NJ, ISBN: 0-877-78298-9.

[54] Nunes, M. B., McPherson, M., Rico, M., (2001). *Constructivist Instructional Design and Development of a Networked Learning Skills (NICLS) Module for Continuing Professional Education Distance Learning*, Proc. ED-MEDIA 2001, pp. 86-91.

[55] Bednar A., Cunningham, D., Duffy, T., Perry, J., (1992) *Theory into practice: how do we link?* In Duffy, T. & Jonassen, D. (Eds.) *Constructivism and the Technology of Instruction: A Conversation*. Lawrence Erlbaum Associates, Mahwah, NJ, ISBN: 0-805-81272-5, pp. 17-34.

[56] Buendia, F., Díaz, P., Benlloch, J.V., (2002). *A Framework for the Instructional Design of Multi-Structured Educational Applications*, Proc. ED-MEDIA 2002, pp. 210-215.

[57] Merrill, M. D. & ID2 Research Team, (1996). *Instructional Transaction Theory: Instructional Design based on Knowledge Objects*. Educational Technology, 36(3), pp. 30-37.

[58] Dobrovolny, J., Spannaus, T., Sims, R., Lamos, J., (2002). *Should instructional designers be project managers?*, Proc. ED-MEDIA 2002, pp. 414-417.

[59] Liu, M., Gibby, S., Quiros, O., Demps, E., (2002). *The Challenge of Being an Instructional Designer for New Media Development: A View From the Practitioners*, Proc. ED-MEDIA 2002, pp. 1151-1157.

[60] Jonassen, D., Mayes, T., Aleese, R. M., (1993). *A manifesto for a constructivist approach to uses of technology in higher education*. In Duffy, T., Lowyck, J., Jonassen, D.,

Welsch, T. M., (Eds.), *Designing Environments for Constructive Learning*. Springer Verlag, Berlin, ISBN: 0-387-56452-7, pp. 231-248.

[61] Scardamalia, M., Bereiter, C., (1996). *Computer support for knowledge building communities*. In Koschmann, T., (Ed.) *CSCL: Theory and practice of an emerging paradigm*. Lawrence Erlbaum Associates, Mahwah, NJ, ISBN: 0-805-81346-2, pp. 249-268.

[62] Eledge Home Page. Online. Eledge Open Learning Management System, University of Utah. Available: http://eledge.sourceforge.net/. 1 January 2004

[63] CSCL2003 - Conference news. Online. Intermedia, University of Bergen. Available: http://www.intermedia.uib.no/cscl/. 1 January 2004.

[64] Microsoft Office - PowerPoint Home Page. Online. Microsoft Corporation. Available: http://www.microsoft.com/office/powerpoint/default.asp. 15 June 2002.

[65] Producer for PowerPoint 2002. Online. Microsoft Corporation. Available: http://www.microsoft.com/office/powerpoint/producer/default.asp. 15 June 2002.

[66] Plug-in Free Internet Delivery of PowerPoint and Video. Online. Impatica, Inc. Available: http://www.impatica.com/. 15 June 2002.

[67] Impatica OnCue: Plug-in Free Internet Delivery of PowerPoint and Video. Online. Impatica, Inc. Available: http://www.impatica.com/imponcue/. 15 November 2003.

[68] WebCT.com. Online. WebCT, Inc. Available: http://www.webct.com/. 15 June 2002.

[69] WebCT authoring system. Online. Neptuno Project. Available: http://www.clt.soton.ac.uk/neptuno/english/ODLlab/develop/p39tools/webct.html. 15 June 2002.

[70] TopClass - WBT Systems - Powering the e-Learning Revolution. Online. WBT Systems. Available: http://www.wbtsystems.com/. 15 June 2002.

[71] TopClass authoring system. Online. Neptuno Project. Available: http://www.clt.soton.ac.uk/neptuno/english/ODLlab/develop/p39tools/topclass.html. 15 June 2002.

[72] Blackboard. Online. Blackboard Inc. Available: http://www.blackboard.com/. 15 June 2002.

[73] Building Courses in Blackboard 5.5. Online. Blackboard Inc. Available: http://products.blackboard.com/gs/training/docs/Bb55_Building_Courses_V2_F1.pdf. 15 June 2002.

[74] Welcome to Mentorware. Online. Mentorware Inc. Available: http://www.mentorware.com/. 15 June 2002.

[75] Apple - Products - HyperCard. Online. Apple Computer, Inc. Available: http://www.apple.com/hypercard/. 15 June 2002.

[76] Apple - HyperCard - Information. Online. Apple Computer, Inc. Available: http://www.apple.com/hypercard/information/. 15 June 2002.

[77] Apple - Software - AppleScript. Online. Apple Computer, Inc. Available: http://www.apple.com/applescript/. 1 December 2003.

[78] Apple Computer, Inc., (1988). *HyperCard script language guide: the HyperTalk language*. Addison Wesley, Macintosh Technical Library. ISBN 0-201-17632-7.

[79] Stand-Alone Code, ad nauseam. Online. Apple Computer, Inc. Available: http://developer.apple.com/technotes/pt/pt_35.html. 1 December 2003.

[80] LiveCard2. Online. Royal Software, Inc. Available: http://www.royalsoftware.com/descriptions/livecardpr2.html. 2 March 2003.

[81] Mentergy - e-Learning Products and Services: Quest Multimedia Authoring. Online. Mentergy, Inc. Available: http://www.mentergy.com/products/authoring_design/quest/60/. 15 June 2002.

[82] Quest authoring system. Online. Neptuno Project. Available: http://www.clt.soton.ac.uk/neptuno/english/ODLlab/develop/p39tools/quest.html. 15 June 2002.

[83] Allen Communication Learning Services - Support: Quest Player Download. Online. Allen Communication Learning Services. Available: http://www.allencomm.com/support/download/player.html. 2 March 2003.

[84] Macromedia - Authorware. Online. Macromedia, Inc. Available: http://www.macromedia.com/software/authorware/. 15 June 2002.

[85] Authorware Professional authoring system. Online. Neptuno Project. Available: http://www.clt.soton.ac.uk/neptuno/english/ODLlab/develop/p39tools/authorware.html. 15 June 2002.

[86] Macromedia - Flash MX. Online. Macromedia, Inc. Available: http://www.macromedia.com/software/flash/. 15 June 2002.

[87] Macromedia - Flash Player. Online. Macromedia, Inc. Available: http://www.macromedia.com/software/flashplayer/. 15 June 2002.

[88] Macromedia - Flash file format (SWF). Online. Macromedia, Inc. Available: http://www.macromedia.com/software/flash/open/licensing/fileformat/license2.html. 15 June 2002.

[89] Macromedia - Flash Player : Nokia Resource Center. Online. Macromedia, Inc. Available: http://www.macromedia.com/software/flashplayer/resources/devices/nokia/. 15 June 2002.

[90] Macromedia Flash - Using ActionScript Overview. Online. Macromedia, Inc. Available: http://www.macromedia.com/support/flash/action_scripts.html. 1 January 2004

[91] Macromedia - Director 8.5 Shockwave Studio. Online. Macromedia, Inc. Available: http://www.macromedia.com/software/director/. 15 June 2002.

[92] Macromedia - Shockwave Player. Online. Macromedia, Inc. Available: http://www.macromedia.com/software/shockwaveplayer/. 15 June 2002.

[93] Macromedia - How to create a Web packaged file. Online, Macromedia, Inc. Available: http://www.macromedia.com/support/authorware/ts/documents/web_package.htm. 2 March 2003.

[94] Macromedia - Xtra Extensions. Online. Macromedia, Inc. Available: http://www.macromedia.com/software/xtras/. 15 December 2003.

[95] Click2learn - Making Knowledge a Tangible Asset. Online. Click2learn, Inc. Available: http://www.click2learn.com/. 15 June 2002.

[96] ToolBook Instructor - Standards-Based Content Authoring. Online. Click2learn, Inc. Available: http://home.click2learn.com/en/toolbook/toolbook_instructor.asp. 15 June 2002.

[97] ToolBook II Instructor authoring system. Online. Neptuno Project. Available: http://www.clt.soton.ac.uk/neptuno/english/ODLlab/develop/p39tools/toolbook.html. 15 June 2002.

[98] Rhodes, J., Bell, C., (2001) *The ToolBook Companion: Solutions, Techniques, Expert Information, and OpenScript Tips*, Platte Canyon Press, ISBN 0-97110-990-7.

[99] Learning Space - Lotus Development Corporation. Online. International Business Machines Corporation. Available: http://www.lotus.com/home.nsf/tabs/learnspace/. 15 June 2002.

[100] FirstClass - Centrinity. Online. Centrinity Inc. Available: http://www.softarc.com/. 15 June 2002.

[101] LearnLinc - Mentergy - Blended e-Learning Solutions. Online. Mentergy Inc. Available: http://www.learnlinc.com/. 15 June 2002.

[102] Mentergy - LearnLinc Feature List. Online. Mentergy Inc. Available: http://www.mentergy.com/products/live_elearning/learnlinc/learnlinc_features.html. 15 June 2002.

[103] Virtual Learning Environments Inc. (VLEI). Online. Virtual Learning Environments Inc. Available: http://www.vlei.com/. 15 June 2002.

[104] What is Virtual-U?. Online. Virtual U Project. Available: http://virtual-u.cs.sfu.ca/vuweb.new/what.html. 15 June 2002.

[105] The Internet CourseReader. Online. TeleLearning National Centre of Excellence (TL-NCE) Virtual U Project, International Labour Organization. Available: http://www.coursereader.net/. 15 June 2002.

[106] SiteScape, Inc. - The Collaboration Company. Online. SiteScape, Inc. Available: http://www.sitescape.com/. 15 June 2002.

[107] Generation21 Learning Systems. Online. Generation21. Available: http://www.gen21.com/. 15 June 2002.

[108] e-Learning solutions: technology to build knowledge online: Pathlore. Online. Pathlore. Available: http://www.pathlore.com/. 15 June 2002.

[109] e-Learning solutions: technology to build knowledge online: Pathlore - Products and Services. Online. Pathlore. Available: http://www.pathlore.com/products_services/virtual_classroom.html. 15 June 2002.

[110] KnowledgePlanet. Online. KnowledgePlanet.com, Inc. Available: http://www.knowledgesoft.com/. 15 June 2002.

[111] VCampus - The e-Learning Application Service .... Online. VCampus Corporation. Available: http://www.vcampus.com/webuol/. 15 June 2002.

[112] Serf Distance Education Environment. Online. Serfsoft Corporation. Available: http://serfsoft.com/. 15 June 2002.

[113] TLM: e-learning applications. Online. TLM Corp. Available: http://thelearningmanager.com/. 15 June 2002.

[114] Question Mark - Getting results. Online. Question Mark Corporation. Available: http://www.questionmark.com/us/home.htm. 15 June 2002.

[115] Welcome to Eloquent. Online. Eloquent, Inc. Available: http://www.eloquent.com/. 15 June 2002.

[116] ChatSpace.com: Home. Online. Akiva. Available: http://www.webboard.ora.com/. 15 June 2002.

[117] Convene. Online. Convene. Available: http://www.convene.com/index.htm. 15 June 2002.

[118] PlaceWare Web Conferencing Provides Live, Interactive Business Meetings and Presentations Over the Internet. Online. PlaceWare. Available: http://www.placeware.com/. 15 June 2002.

[119] Ucompass. Online. Ucompass.com, Inc. Available: http://www.ucompass.com/. 15 June 2002.

[120] Interwise. Online. Interwise. Available: http://www.interwise.com/. 15 June 2002.

[121] Home. Online. e-com inc. Available: http://www.theorix.com/. 15 June 2002.

[122] AICC - Aviation Industry CBT Committee. Online. AICC. Available: h tp://www.aicc.org/. 15 June 2002.

[123] Understanding AICC "Compliance". Online. AICC. Available: http://www.aicc.org/pages/primer.html. 15 June 2002.

[124] Advanced Distributed Learning Network (ADLNet) - SCORM Overview. Online. Advanced Distributed Learning. Available: http://www.adlnet.org/index.cfm?fuseaction=scormabt. 15 June 2002.

[125] Advanced Distributed Learning Network (ADLNet). Online. Advanced Distributed Learning. Available: h p://www.adlnet.org/. 15 June 2002.

[126] E-learning: Education Is One Of The Applications Still Growing After Dot.com Meltdown :: Distance-Educator.com's Daily News :: Distance education news from around the world! Online. Distance-Educator.com, Inc. Available: http://www.distance-educator.com/dnews/Article7555.phtml. I December 2003.

Software Engineering Resources

[127] Gamma, E., Helm, R., Johnson, R., Vlissides, J., (1994). *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison Wesley Professional Computing Series, ISBN 0-201-63361-2.

[128] Raskin, J., (2000). *The Humane Interface: New Directions for Designing Interactive Systems*, Addison-Wesley, First Edition, ISBN: 0-201-37937-6.

[129] Mayhew, D. J., (1997). *Principles and Guidelines in Software User Interface Design*, Pearson Education POD, First Edition, ISBN: 0-137-21929-6.

[130] Smith, S. L., Mosier, J. N., (1986). *Guidelines for Designing User Interface Software*. The MITRE Corporation ESD-TR-86-278, Bedford, MA, ISBN: 9-992-08041-8.

[131] Cooper, A., (1999). *The inmates are running the asylum*. SAMS, Indianapolis, ISBN: 0-672-31649-8.

[132] Brooks, F. P., (1986). "No silver bullet—essence and accidents of software engineering", in *Information Processing 86*, Kluger, H. J. (Ed.) Elsevier Science, Amsterdam, North Holland, pp. 1069-1076.

[133] Brooks, F. P., (1995). *The mythical man-month: essays on software engineering*, 20th Anniversary Edition, Addison Wesley Longman, ISBN 0-201-83595-9, pp. 182.

[134] ISO/IEC 14882, (1998). *Information Technology — Programming Languages — C++*, International Standards Organisation.

[135] Tucker, A. B. Jr. et al., (Eds.), (1997). *The Computer Science and Engineering Handbook*. CRC Press in cooperation with ACM, ISBN 0-8493-2909-4.

[136] Cargill, T.A., (1991). *Controversy: The case against multiple inheritance in C++*. Computing Systems 4, 1, pp. 69-82.

[137] ISO 9241, (1996). *Ergonomic Requirements For Visual Display Terminals*, International Standards Organisation.

[138] ISO/IEC 9075-1, (1999). *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*, International Standards Organisation.

[139] IEEE Std 1471-2000, (2000). *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, Institute of Electrical and Electronics Engineers, ISBN 0-7381-2518-0.

[140] Shannon, C. E., (1948). *A mathematical theory of communication*, Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July and October, 1948. Also available online: http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html. 15 June 2002

[141] Patterson, D. A., Hennessy, J. L., (2003). *Computer Architecture. A Quantitative Approach*, Morgan Kaufman Publishers, San Mateo, CA, Third Edition, ISBN 1-55860-724-2.

[142] Coplien, J. O., (1992). *Advanced C++ Programming Styles and Idioms*. Addison-Wesley, Reading, MA.

[143] Musser, D. R., Saini A., (2001). *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*, Addison-Wesley Professional Computing Series, ISBN 0-201-63398-1.

[144] Alexandrescu, A., (2001). *Modern C++ Design*, Addison-Wesley The C++ In-Depth Series, ISBN 0-201-70431-5, pp. 99-128.

[145] UML Home Page. Online. Object Management Group, Inc. Available: http://www.uml.org/. 1 January 2004

[146] Rational Unified Process - Product Overview - IBM Software. Online. IBM Corporation. Available: http://www-3.ibm.com/software/awdtools/rup/. 15 November 2003.

[147] CodeWarrior Desktop Products. Online. Metrowerks, a Motorola company. Available: http://www.metrowerks.com/MW/Develop/Desktop/default.htm. 1 January 2004.

[148] The Metrowerks PowerPlant framework. Online. Metrowerks. Available: http://www.metrowerks.com/desktop/mactools/powerplant/. 15 June 2002.

[149] The Metrowerks PowerPlant whitepaper. Online. Metrowerks. Available: http://www.metrowerks.com/whitepapers/powerplant/. 15 June 2002.

[150] Microsoft Visual C++ .NET Home Page. Online. Microsoft Corporation. Available: http://msdn.microsoft.com/visualc/. 15 June 2002.

[151] Microsoft Foundation Class Library. Online. Microsoft Corporation. Available: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcmfc98/html/mfchm.asp. 15 November 2003.

[152] Dynamic Link Library - a searchWin2000 definition - see also: DLL. Online. TechTarget. Available: http://searchwin2000.techtarget.com/sDefinition/0,,sid1_gci213902,00.html. 23 December 2003.

[153] DDE - a whatis definition - see also: Dynamic Data Exchange. Online. TechTarget. Available: http://whatis.techtarget.com/definition/0,,sid9_gci213884,00.html. 23 December 2003.

[154] ActiveX - a searchWin2000 definition, Online. TechTarget. Available: http://searchwin2000.techtarget.com/sDefinition/0,,sid1_gci211521,00.html. 23 December 2003.

[155] Cocoa. Online. Apple Computer, Inc. Available: http://developer.apple.com/cocoa/. 1 December 2003.

[156] Component Manager (IM: MTb). Online. Apple Computer, Inc. Available: http://developer.apple.com/documentation/mac/MoreToolbox/MoreToolbox-333.html. 30 November 2003.

[157] Sound Manager (IM: S). Online. Apple Computer, Inc. Available: http://developer.apple.com/documentation/mac/Sound/Sound-44.html. 1 January 2004.

[158] Apple Computer, Inc., (1992) *Apple Macintosh Human Interface Guidelines*. Addison-Wesley Publishing Company, Reading, MA, ISBN: 0-201-62216-5.

[159] Apple - Support - Mac OS 9. Online. Apple Computer, Inc. Available: http://www.info.apple.com/usen/macos9/. 1 January 2004.

[160] Apple - Mac OS X. Online. Apple Computer, Inc. Available: http://www.apple.com/macosx/. 15 June 2002.

[161] Carbon. Online. Apple Computer, Inc. Available: http://developer.apple.com/carbon/. 15 June 2002.

[162] Print2Pict 3.7.1 – VersionTracker. Online TechTracker, Inc. Available: http://www.versiontracker.com/dyn/moreinfo/macos/3584&vid=3593. 1 December 2003.

[163] The World Wide Web Consortium. Online. W3C. Available: http://www.w3c.org/. 15 June 2002.

[164] Unicode Home Page. Online. Unicode, Inc. Available: http://www.unicode.org/. 15 June 2002.

[165] ASCII Table - ASCII character codes and html, octal, hex and decimal charts. Online. ASCIITable.com Available: http://www.asciitable.com/. 30 November 2003.

[166] ISO 8879, (1986). *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*, International Standards Organisation.

[167] Extensible Markup Language (XML). Online. W3C. Available: http://www.w3.org/XML/. 15 June 2002.

[168] XML in 10 points. Online. W3C. Available: http://www.w3.org/XML/1999/XML-in-10-points. 15 June 2002.

[169] Extensible Markup Language (XML) 1.0 (Second Edition). Online. W3C. Available: http://www.w3.org/TR/REC-xml. 15 June 2002.

[170] Namespaces in XML. Online. W3C. Available: http://www.w3.org/TR/REC-xml-names/. 15 June 2002.

[171] Berners-Lee, T., Hendler, J., Lassila, O., (2001). *The Semantic Web*. Scientific American, May 2001, pp. 35-43. Also online. Scientific American. Available. http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2.1 December 2003.

[172] W3C Semantic Web. Online. W3C. Available: http://www.w3.org/2001/sw/. 15 June 2002.

[173] Resource Description Framework (RDF) / W3C Semantic Web Activity. Online. W3C. Available: http://www.w3.org/RDF/. 15 June 2002.

[174] XML Linking Language (XLink) Version 1.0. Online. W3C. Available: http://www.w3.org/TR/xlink/. 15 June 2002.

[175] XML Path Language (XPath). Online. W3C. Available: http://www.w3.org/TR/xpath. 15 June 2002.

[176] XML Pointer Language (XPointer) Version 1.0. Online. W3C. Available: http://www.w3.org/TR/WD-xptr. 15 June 2002.

[177] The Extensible Stylesheet Language (XSL). Online. W3C. Available: http://www.w3.org/Style/XSL/. 15 June 2002.

[178] XSL Transformations (XSLT). Online. W3C. Available: http://www.w3.org/TR/xslt. 15 June 2002.

[179] W3C Document Object Model. Online. W3C. Available: http://www.w3.org/DOM/. 15 June 2002.

[180] W3C XML Schema. Online. W3C. Available: http://www.w3.org/XML/Schema. 15 June 2002.

[181] Web Content Accessibility Guidelines 1.0. Online. W3C. Available: http://www.w3.org/TR/WAI-WEBCONTENT/. 1 January 2004.

[182] Web Content Accessibility Guidelines 2.0. Online. W3C. Available: http://www.w3.org/WAI/GL/WCAG20/. 1 January 2004.

[183] File Transfer Protocol. Online. Internet Engineering Task Force, Network Working Group. Available: http://www.ietf.org/rfc/rfc0959.txt. 1 January 2004.

[184] RFC 2616: Hypertext Transfer Protocol — HTTP/1.1. Online. Internet Engineering Task Force, Network Working Group. Available: http://www.ietf.org/rfc/rfc2616.txt. 15 June 2002.

[185] Scalable Vector Graphics (SVG). Online. W3C. Available: http://www.w3.org/Graphics/SVG/Overview.htm8. 15 June 2002.

[186] Scalable Vector Graphics (SVG) 1.0 Specification. Online. W3C. Available: http://www.w3.org/TR/SVG/. 15 June 2002.

[187] Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. W3C Recommendation 15-June-1998. Online. W3C. Available: http://www.w3.org/TR/REC-smil/. 15 June 2002.

[188] Synchronized Multimedia Integration Language (SMIL 2.0). W3C Recommendation 07 August 2001. Online. W3C. Available: http://www.w3.org/TR/smil20. 15 June 2002.

[189] Web3D CONSORTIUM | Specifications | X3D. Online. Web3D Consortium. Available: http://www.web3d.org/x3d.html. 15 June 2002.

[190] HTML Home Page. Online. W3C. Available: http://www.w3.org/MarkUp/. 15 June 2002.

[191] HTML 4.01 Specification. Online. W3C. Available: http://www.w3.org/TR/html4/. 15 June 2002.

[192] XHTML 1.0: The Extensible HyperText Markup Language. Online. W3C. Available: http://www.w3.org/TR/xhtml1/. 15 June 2002.

[193] XHTML 1.1 - Module-based XHTML. Online. W3C. Available: http://www.w3.org/TR/xhtml11/. 15 June 2002.

[194] Cascading Style Sheets. Online. W3C. Available: http://www.w3.org/Style/CSS/. 15 June 2002.

[195] XHTML Modularization Overview. Online. W3C. Available: http://www.w3.org/MarkUp/modularization. 15 June 2002.

[196] Uniform Resource Identifiers (URI): Generic Syntax. Online. Internet Engineering Task Force, Network Working Group. Available: http://www.ietf.org/rfc/rfc2396.txt. 15 June 2002.

[197] Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. Online. Internet Engineering Task Force, Network Working Group. Available: http://www.ietf.org/rfc/rfc2045.txt. 1 January 2004.

[198] IANA | Application Media-Types. Online. Internet Assigned Numbers Authority. Available: http://www.iana.org/assignments/media-types/application/. 1 January 2004.

[199] Adobe Systems Incorporated, (2001), *PDF Reference (Third Edition) version 1.4*, Addison-Wesley, ISBN 0-201-75839-3. Also available online: http://partners.adobe.com/asn/developer/acrosdk/docs/filefmtspecs/PDFReference.pdf. 15 June 2002.

[200] Adobe Reader. Online. Adobe Systems Incorporated. Available: http://www.adobe.com/products/acrobat/readermain.html. 1 January 2004.

[201] Adobe Acrobat Distiller Server. Online. Adobe Systems Incorporated. Available: http://www.adobe.com/products/acrdis/main.html. 15 June 2002.

[202] Adobe Acrobat Distiller API Reference. Online. Adobe Systems Incorporated. Available: http://partners.adobe.com/asn/developer/acrosdk/docs/createpdfapi/DistillerAPIReference.pdf. 15 June 2002.

[203] pdfmark Reference Manual. Online. Adobe Systems Incorporated. Available: http://partners.adobe.com/asn/developer/acrosdk/docs/createpdfapi/pdfmarkReference.pdf. 15 June 2002.

[204] Adobe Acrobat PDFWriter API Reference. Online. Adobe Systems Incorporated. Available: http://partners.adobe.com/asn/developer/acrosdk/docs/createpdfapi/PDFWriterAPIReference.pdf. 15 June 2002.

[205] Refactoring Home Page. Online. Martin Fowler. Available: http://www.refactoring.com/. 15 June 2002.

[206] Alpha list of Refactorings. Online. Martin Fowler. Available: http://www.refactoring.com/catalog/index.html. 15 June 2002.

[207] The UNIX System, UNIX System. Online. The Open Group. Available: http://www.unix-systems.org/. 30 November 2003.

[208] Java Technology. Online. Sun Microsystems, Inc. Available: http://java.sun.com/. 1 January 2004.

[209] Applets. Online. Sun Microsystems, Inc. Available: http://java.sun.com/applets/. 1 January 2004.

[210] Java 2 Platform, Enterprise Edition (J2EE) Online. Sun Microsystems, Inc. Available: http://java.sun.com/j2ee/. 30 November 2003.

[211] Unisys | LZW Patent and Software Information. Online. Unisys Corporation. Available: http://www.unisys.com/about__unisys/lzw. 1 January 2004

[212] Oracle Corporation. Online. Oracle Corporation. Available: http://www.oracle.com/. 30 November 2003.

Data Compression Resources

[213] V.90 Modem Standard. Online. Copper Pair Communications, Inc. Available: http://www.v90.com/. 15 June 2002.

[214] Data communication over the telephone network. Online. International Telecommunication Union. Available: http://www.itu.int/rec/recommendation.asp?type=products&lang=e&parent=T-REC-V. 15 June 2002.

[215] ITU-T Recommendation T.800 | ISO/IEC 15444-1, (2000). *JPEG 2000 Image Coding System: Core Coding System*, International Standards Organisation.

[216] JPEG JFIF. Online. W3C. Available: http://www.w3.org/Graphics/JPEG/. 15 June 2002.

[217] Independent JPEG Group. Online. Independent JPEG Group. Available: http://www.ijg.org/. 15 June 2002.

[218] PNG (Portable Network Graphics) Home Site. Online. Greg Roelofs. Available: http://www.libpng.org/pub/png/. 15 June 2002.

[219] Intro to PNG Features. Online. Greg Roelofs. Available: http://www.libpng.org/pub/png/pngintro.html. 15 June 2002.

[220] PNG Specification: Introduction. Online. Greg Roelofs. Available: http://www.libpng.org/pub/png/spec/PNG-Introduction.html. 15 June 2002.

[221] PNG Specification: Chunk Specifications. Online. Greg Roelofs. Available: http://www.libpng.org/pub/png/spec/PNG-Chunks.html#C.IHDR. 15 June 2002.

[222] pngquant Home Page. Online. Greg Roelofs. Available: http://www.libpng.org/pub/png/apps/pngquant.html. 15 June 2002.

[223] MNG (Multiple-image Network Graphics) Home Page. Online. Greg Roelofs. Available: http://www.libpng.org/pub/mng/. 30 November 2003.

[224] zlib Home Site. Online. Greg Roelofs and Jean-loup Gailly. Available: http://www.gzip.org/zlib/. 15 June 2002.

[225] Graphics Interchange Format(sm). Version 89a. Online. W3C. Available: http://www.w3.org/Graphics/GIF/spec-gif89a.txt. 15 June 2002.

[226] TIFF 6.0 Specification. Online. Adobe Systems Incorporated. Available: http://partners.adobe.com/asn/developer/PDFS/TN/TIFF6.pdf. 15 June 2002.

[227] GSM 06.10 lossy speech compression library website. Online. jutta@pobox.com. Available: http://kbs.cs.tu-berlin.de/~jutta/toast.html. 15 June 2002.

[228] Apple – QuickTime. Online. Apple Computer, Inc. Available: http://www.apple.com/quicktime/specifications.html. 15 June 2002.

[229] The Apple QuickTime technology documentation. Online. Apple Computer, Inc. Available: http://developer.apple.com/techpubs/quicktime/quicktime.html. 15 June 2002.

[230] Apple – Products – QuickTime and SMIL. Online. Apple Computer, Inc. Available: http://www.apple.com/quicktime/authoring/qtsmil.html. 15 June 2002.

[231] Apple – QuickTime – QuickTime VR Authoring. Online. Apple Computer, Inc. Available: http://www.apple.com/quicktime/qtvr/. 1 March 2003.

[232] Apple – QuickTime – Download. Online. Apple Computer, Inc. Available: http://www.apple.com/quicktime/download/. 30 November 2003.

[233] Audio File Formats FAQ: File Formats. http://sox.sourceforge.net/AudioFormats-11.html. 26 December 2003.

[234] ITU-T G.711, (1988). *Pulse code modulation (PCM) of voice frequencies*. International Telecommunications Union. Also available online: http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-G.711. 1 January 2004.

[235] ITU-T G.726, (1990). *40, 32, 24, 16 kbit/s adaptive differential pulse code modulation (ADPCM)*. International Telecommunications Union. Also available online: http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-G.726. 1 January 2004.

[236] ITU-T G.723, (1988). *Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s*. International Telecommunications Union. Also available online: http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-G.723.1. 26 December 2003.

[237] ITU-T H.323, (2003). *Packet-based multimedia communications systems*. International Telecommunications Union. Also available online: http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.323. 1 January 2004.

[238] ITU-T H 324, (2002). *Terminal for low bit-rate multimedia communication*. International Telecommunications Union. Also available online: http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.324. 1 January 2004.

[239] Consequences of Nyquist Theorem for Acoustic Signals Stored in Digital Format. Online. Digital Recordings. Available: http://www.digital-recordings.com/publ/pubneq.html. 15 June 2002.

[240] draft-ema-vpim-msgsm-01 - ALterNIC - Network Information Center - Diane Boling. Online. KBLabs. Available: http://alternic.net/drafts/drafts-e-f/draft-ema-vpim-msgsm-01.html. 15 June 2002.

[241] Degener, J., (1994). *Digital Speech Compression, Putting the GSM 06.10 RPE-LTP algorithm to work*, Dr. Dobb's Journal, December 1994. Also available online: http://www.ddj.com/documents/s=1012/ddj9412b/9412b.htm. 15 June 2002.

[242] xiph.org: Ogg Vorbis. Online. xiph.org Available: http://www.xiph.org/ogg/vorbis/index.html. 15 June 2002.

[243] DSP Group | Products & Technologies. Online. DSP Group. Available: http://www.dspg.com/prodtech/truespch/6353.htm. 15 June 2002.

[244] MPEG Home Page. Online. Moving Picture Experts Group. Available: http://www.chiariglione.org/mpeg/index.htm. 15 June 2002.

[245] ISO/IEC 11172, (1993-1999). *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s*, International Standards Organisation. In 5 parts, as described online: http://www.chiariglione.org/mpeg/standards/mpeg-1/mpeg-1.htm. 1 January 2004.

[246] ISO/IEC 13818, (1998-2000). *Information technology - Generic coding of moving pictures and associated audio information*, International Standards Organisation. In 9 parts, as described online: http://www.chiariglione.org/mpeg/standards/mpeg-2/mpeg-2.htm. 1 January 2004.

[247] DVD Forum. Online. DVD Forum. Available: http://www.dvdforum.org/forum.shtml. 1 January 2004.

[248] Fraunhofer IIS - Audio & Multimedia - MPEG Audio Layer-3. Online. Fraunhofer-Gesellschaft. Available: http://www.iis.fraunhofer.de/amm/techinf/layer3/. 1 January 2004

[249] MPEG-4 description. Online. Moving Picture Experts Group. Available: http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm. 1 January 2004.

[250] MPEG-7 overview. Online. Moving Picture Experts Group. Available: http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm. 1 January 2004.

[251] ISO/IEC 14772, (1997) *VRML97 Specification*, International Standards Organisation. Also available online. Web3D Consortium. Available: http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm. 15 June 2002.

[252] Fraunhofer IIS-A - AMM - Layer-3 Info. Online. Fraunhofer-Gesellschaft. Available: http://www.iis.fhg.de/amm/techinf/layer3/index.html. 15 June 2002.

[253] mp3licensing.com – Home. Online. Thomson Multimedia. Available. http://www.mp3licensing.com/. 15 June 2002.

[254] Advanced Audio Coding. Online. Dolby Laboratories, Inc. Available. http://www.aac-audio.com/. 15 June 2002.

[255] TwinVQ. Online. TwinVQ.org Available. http://www.twinvq.org/english/index_en.html. 15 June 2002.

[256] Windows Media Technologies Home. Online. Microsoft Corporation. Available. http://www.microsoft.com/windows/windowsmedia/default.asp. 15 June 2002.

[257] Windows Media Format. Online. Microsoft Corporation. Available: http://www.microsoft.com/windows/windowsmedia/format/default.aspx. 26 December 2003.

[258] Lucent Digital Radio announces new product line based on the Lucent Perceptual Audio Coder (PAC™). Online. Lucent Technologies. Available. http://www.lucent.com/press/0499/990412.cob.html. 15 June 2002.

[259] Sound Compression and Expansion (IM: S). Online. Apple Computer, Inc., 1996. Available: http://developer.apple.com/documentation/mac/Sound/Sound-49.html. 1 January 2004

[260] QuickTime File Format: Formats Not Currently in Use: MACE 3:1 and 6:1. Online. Apple Computer, Inc. Available: http://developer.apple.com/documentation/QuickTime/QTFF/QTFFChap3/chapter_4_section_13.html. 1 January 2004

General Resources

[261] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S., (1977). *A Pattern Language*, Oxford University Press, New York, ISBN: 0-195-01919-9, pp. x^{l}.

[262] Kurzweil, R., (1999). *The age of spiritual machines*, Phoenix, ISBN:0-75380-767-X.

[263] Ecriture. Online. ANKH, Revue d'Egyptologie et des Civilisations africaines. Available: http://www.ankhonline.com/ecriture1.htm. 15 June 2002.

[264] Some Dates in the History of Communication Technologies. Online. Thistlerose Publications. Available: http://worldhistorysite.com/culttech.html. 15 June 2002.

[265] Ascher, M., Ascher, R., (1997). *Mathematics of the Incas : Code of the Quipu*, Dover Publications, New York, ISBN: 0-486-29554-0, pp. 29-31.

[266] John Chambers :: Home, Online. Cisco Systems, Inc., Available: http://newsroom.cisco.com/dlls/tln/exec_team/chambers/. 1 December 2003.

[267] Myst. Online. Cyan. Available: http://sirrus.cyan.com/Online/Myst/MystHome. 15 June 2002.

[268] Moores Law - Webopedia.com. Online. INT Media Group, Incorporated. Available: http://www.webopedia.com/TERM/M/Moores_Law.html. 15 June 2002.

[269] Intel Corporation - Welcome to Intel.com. Online. Intel Corporation. Available: http://www.intel.com/. 15 June 2002.

[270] Bass, M. J., Christensen, C. M., (2002). *The Future of the Microprocessor Business*, IEEE Spectrum, April 2002, Vol. 39(4), ISSN: 0018-9235, pp. 34-49.

[271] NCAM/Captioner for Flash. Online. National Center for Accessible Media. Available: http://ncam.wgbh.org/webaccess/magpie/magpie2_captioner.html. 1 January 2004.

[272] Flash News Flash: It's Accessible. Online. Lycos, Inc. Available: http://www.wired.com/news/culture/0,1284,51638,00.html. 15 June 2002.

[273] NCAM/Media Access Generator (MAGpie). Online. National Centre for Accessible Media. Available: http://ncam.wgbh.org/webaccess/magpie/. 15 June 2002.

[274] Diskeeper - Defragment Your Hard Drive. Online. Executive Software International. Available: http://www.executivesoftware.com/diskeeper/diskeeper.asp. 15 June 2002.

[275] Microsoft Office. Online. Microsoft Corporation. Available: http://www.microsoft.com/office/word. 1 January 2004.

[1] Note that the page number in this instance is the Roman numeral X.

[276] Microsoft Office. Online. Microsoft Corporation. Available: http://www.microsoft.com/office/excel. 1 January 2004.

[277] Microsoft Office. Online. Microsoft Corporation. Available: http://www.microsoft.com/frontpage. 1 January 2004.

[278] WordPerfect Office - Product Information. Online. Corel Corporation. Available: http://www.wordpe fect.com. 1 January 2004.

[279] IBM Lotus Software - SmartSuite. Online. International Business Machines Corporation. Available: http://www.lotus.com/products/smrtsuite.nsf/wPages/wordpro. 1 January 2004.

[280] IBM Lotus Software - SmartSuite. Online. International Business Machines Corporation. Available: http://www.lotus.com/products/smr suite.nsf/wPages/freelance. 1 January 2004.

[281] IBM Lotus Software - SmartSuite. Online. International Business Machines Corporation. Available: http://www.lotus.com/products/smrtsuite.nsf/wPages/123. 1 January 2004.

[282] Acrobat family. Online. Adobe Systems Incorporated. Available: http://www.adobe.com/products/acrobat/main.html. 1 January 2004.

[283] Adobe Photoshop. Online. Adobe Systems Incorporated. Available: http://www.adobe.com/products/photoshop/main.html. I January 2004.

[284] Adobe Premiere. Online. Adobe Systems Incorporated. Available: http://www.adobe.com/products/premiere/main.html. 1 January 2004.

[285] Macromedia - Dreamweaver MX 2004. Online. Macromedia, Inc. Available: http://www.macromedia.com/software/dreamweaver/. 1 January 2004.

[286] Macromedia - Products : HomeSite 5.5. Online. Macromedia, Inc. Available: http://www.macromedia.com/software/homesite/. 1 January 2004.

[287] Netscape Composer. Online. Netscape. Available: http://wp.netscape.com/communicator/composer/v4.0/index.html. 1 January 2004.

[288] Jasc Paint Shop Pro 8. Online. Jasc Software, Inc. Available: http://www.jasc.com/products/paintshoppro/. 1 January 2004.

[289] Apple - iLife - iMovie. Online. Apple Computer, Inc. Available: http://www.apple.com/imovie/. 1 January 2004.

[290] Apple - Final Cut Pro 4. Online. Apple Computer, Inc. Available: http://www.apple.com/finalcutpro/. I January 2004.

[291] Sony Pictures Digital - The Sound Forge Product Family. Online. Sony Pictures Digital Inc. Available: http://mediasoftware.sonypictures.com/products/soundforgefamily.asp. 1 January 2004.

[292] LaTeX Project Home page. Online. LaTeX3 Project. Available: http://www.latex-project.org/. 5 October 2003.

[293] The Apache Cocoon Project. Online. The Apache Software Foundation. Available: http://cocoon.apache.org/. 1 December 2003.

[294] OceanESIP Overview. Online. Jet Propulsion Laboratory, California Institute of Technology. Available: http://oceanesip.jpl.nasa.gov/. 30 November 2003.