Crossroads: A Collaborative Design Notebook
For Constructionist Learning Activities

by

Oludotun A. Fashoyin

Submitted to the
Department of Electrical Engineering and Computer Science on
May 23, 2001

in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering and
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The tools and technologies developed in the Epistemology and Learning Group at the MIT Media Laboratory are used everyday in new and exciting ways by children and adults from a wide range of backgrounds in a growing number of settings, both in and out of schools. There is a great opportunity to bring together a strong community of learners, educators, and learning researchers to talk about and reflect on their experiences and practices. This research is intended to facilitate the formation of this community by providing a set of tools that support collaboration and communication among its members. Crossroads is designed to enable people (particularly children) to work together not only to tell others about their thoughts, ideas, and what they create to make these thoughts and ideas concrete and shareable, but also to get feedback and reflect on their ongoing projects. In this thesis, we present the design, implementation, preliminary evaluation, and future development directions of Crossroads, a collection of easy-to-use authoring tools – that support text, image, video, and audio – and simple administrative tools – that make setting up, customizing, and maintaining new sites with different work and connectivity requirements easy.

Thesis Supervisor: Bakhtiar Mikhak
Title: Research Scientist, MIT Media Laboratory
To Novice, for giving me a much-appreciated headstart...
# TABLE OF CONTENTS

1 INTRODUCTION ............................................................................................................. 7
2 RELATED WORK ........................................................................................................... 7
   2.1 QuickPlace™ ......................................................................................................... 16
   2.2 JavaCAP ............................................................................................................... 20
   2.3 Belvedere ............................................................................................................. 26
   2.4 Closing Remarks .................................................................................................. 32
3 SCENARIOS .................................................................................................................. 33
4 DEVELOPMENT .......................................................................................................... 42
   4.1 Evolution .............................................................................................................. 42
   4.2 Design .................................................................................................................. 44
      4.2.1 Architecture ................................................................................................... 45
      4.2.2 Client Applications ....................................................................................... 46
         4.2.2.1 Editor ....................................................................................................... 47
         4.2.2.2 Administrator ......................................................................................... 51
         4.2.2.3 Resource Folder ..................................................................................... 55
         4.2.2.4 Project Browser ..................................................................................... 58
      4.2.3 Server Applications ...................................................................................... 62
         4.2.3.1 Standard Server ...................................................................................... 62
         4.2.3.2 Administration Server ........................................................................... 64
   4.3 Implementation ...................................................................................................... 65
      4.3.1 Client Applications ....................................................................................... 65
         4.3.1.1 Editor ....................................................................................................... 65
         4.3.1.2 Administrator ......................................................................................... 75
         4.3.1.3 Resource Folder ..................................................................................... 80
         4.3.1.4 Project Browser ..................................................................................... 82
         4.3.1.5 Failures .................................................................................................... 84
         4.3.1.6 Performance ............................................................................................. 84
         4.3.1.7 Potential Modifications ........................................................................... 85
      4.3.2 Server Applications ....................................................................................... 85
         4.3.2.1 Standard Server ....................................................................................... 85
         4.3.2.2 Administration Server ........................................................................... 86
         4.3.2.3 Failures .................................................................................................... 86
         4.3.2.4 Performance ............................................................................................. 86
      4.3.3 Resources Used .............................................................................................. 86
5 PRELIMINARY EVALUATION .................................................................................... 89
6 FUTURE WORK ......................................................................................................... 92
7 CONCLUSION ............................................................................................................ 100
ACKNOWLEDGEMENTS ............................................................................................... 102
BIBLIOGRAPHY ............................................................................................................ 103
APPENDIX .................................................................................................................... 105
A Application Abstractions ........................................................................................... 105
   A.1 Editor abstractions .............................................................................................. 105
      A.1.1 CrossroadsEditor ......................................................................................... 105
A.1.2 FileManager ................................................................. 106
A.1.3 EditManager ............................................................... 106
A.1.4 MediaManager ........................................................... 106
A.1.5 ToolsManager ............................................................ 107
A.2 Administrator abstractions ............................................ 107
A.2.1 CrossroadsAdministrator .......................................... 107
A.2.2 FileManager ............................................................... 107
A.2.3 ToolsManager ............................................................ 108
A.3 Resource Folder abstractions ......................................... 108
A.3.1 CrossroadsResourceFolder ........................................ 108
A.4 Project Browser abstractions .......................................... 108
A.4.1 CrossroadsProjectBrowser ....................................... 108
A.5 Standard Server abstractions ......................................... 109
A.5.1 CrossroadsStandardServer ....................................... 109
A.6 Administration Server abstractions ............................... 109
A.6.1 CrossroadsAdministrationServer .............................. 109
B Utility Abstractions ........................................................ 109
B.1 Core data abstractions .................................................. 109
B.1.1 User ....................................................................... 109
B.1.2 Administrator .......................................................... 110
B.1.3 Session ................................................................. 110
B.1.4 Activity ................................................................. 110
B.1.5 Team ..................................................................... 111
B.1.6 Project ................................................................. 111
B.1.7 ProjectVersion ......................................................... 111
B.1.8 MediaContent .......................................................... 112
B.1.9 MediaSource .......................................................... 112
B.1.10 MediaContext .......................................................... 112
B.1.11 MediaResource ........................................................ 113
B.2 Server data abstractions ............................................... 113
B.2.1 DatabaseAccess ......................................................... 113
B.2.2 AdminDatabaseAccess ............................................. 115
B.3 Network connection abstractions .................................. 115
B.3.1 ClientConnection ...................................................... 115
B.3.2 ServerConnection ..................................................... 116
B.3.3 AdminServerConnection ......................................... 117
B.4 Content presentation abstractions ................................ 117
B.4.1 MediaPanel ............................................................ 117
B.4.2 MediaComponent .................................................... 118
# LIST OF FIGURES

1. Designating access to documents in QuickPlace™ .................................................. 17
2. Creating an inner room in QuickPlace™ ................................................................. 18
3. Authoring content in QuickPlace™ ......................................................................... 19
4. The “Problem Presentation” Scene in JavaCAP ...................................................... 23
5. The “Alternatives Selection” Scene in JavaCAP ..................................................... 24
6. The inquiry diagramming tool in Belvedere ............................................................ 30
7. Ms. Wilson using the Crossroads Administrator to add Leann and Leroy to Team 2 .......................................................... 36
8. Leroy using the Crossroads Editor to create sample projects .................................. 38
9. The architecture of Crossroads ................................................................................ 46
10. The module dependency diagram for the Crossroads Editor ................................... 50
11. The module dependency diagram for the Crossroads Administrator ..................... 54
12. The module dependency diagram for the Crossroads Resource Folder .................. 58
13. The module dependency diagram for the Crossroads Project Browser ................... 61
14. The module dependency diagram for the Crossroads Standard Server .................. 63
15. The module dependency diagram for the Crossroads Administration Server .......... 65
1 INTRODUCTION

In recent decades, there has been a growing emphasis on supporting collaboration among groups of people in many areas of life, in particular work and school. The rate of technological advancements in networking over the past two decades – which are reshaping how we think about, discuss, and support collaboration – is one of the significant factors that are fueling this growth. In a little over a quarter of a century, a number of new means of communication have become ubiquitous in the developed world.

Starting in late seventies in a handful of research institutions in United States and Europe, and continuing into the early eighties in more and more universities, networks such as BITNET provided students, faculty, and researchers a far more convenient way to distribute information to their colleagues and communities of interest than traditional paper copies. Furthermore, the emergence of the newsgroups and electronic mail systems a few years later helped remove many communication barriers as these tools began to become accessible and attractive to the general public. People could send electronic messages to each other regardless of location, and companies could make transactions with other companies in a matter of minutes.

As the World Wide Web started becoming popular in the early 1990's, collaboration support was reaching new heights. The increasing ease of Internet connectivity and the development of the browser, the Web’s powerful and easy-to-use “killer” application, transformed the once relatively unknown Internet into an extensive medium for various kinds of collaboration over continental distances. Furthermore, with the budding interest in peer-to-peer networks, it becomes difficult to disregard the multitude of benefits that collaboration using computer technology has on society today.
One societal area in which collaboration plays an integral part is the workplace. This observation can be attributed to the fact that companies increasingly work in teams. From the top executives who meet regularly to make decisions and discuss the progress of the company to the employees who develop products and services, the company as a whole is working together to achieve a common set of goals.

In this day and age, as technological developments continue to grow exponentially, much interest has been devoted to the implications of computer technologies in the workplace, particularly in facilitating team collaboration. Of worthy mention are the conferences on computer supported cooperative work (CSCW) [5,6]. These conferences serve as forums for prominent researchers, developers, and other practitioners in educational and business institutions to address issues concerning the use of computer technologies to support collaborative activities. They also help to facilitate engaging discussions on the impact of digital collaboration technologies on users, groups, organizations and society. These conferences are structured in terms of workshops on various topics, including infrastructures for collaboration, organizational culture, shared visual spaces, and web-based customer service.

With increasing Web use, many tools designed to support collaboration in the workplace have adopted the World Wide Web as their host platform. A good example of a tool with these properties is QuickPlace™ [23]. A popular software application developed by the Lotus® Development Corporation, QuickPlace™ allows team members to manage a shared workspace on the Web. It also boasts several communication capabilities including e-mail, chat services, newsletters, and bulletin boards, and other useful features, e.g., a calendar. Chapter 2 includes a more detailed discussion on QuickPlace™.
Collaborative learning and work are also increasingly recognized and discussed as fundamental components of our educational institutions. In several classroom activities in K12 schools, students are more frequently divided into teams in order to accomplish a particular task. In addition, teachers and other educators often work together to design activities for their students as part of the school curriculum. In other education-based settings such as after-school centers and museums, students are also assigned team projects, which help to facilitate their learning, both individually and collaboratively.

Interest in the potential impact of technology on collaborative learning has been on rise recently, in a manner quite similar to the integration of technology into collaborative work that was discussed earlier. This interest has been promoted specifically by efforts such as the conference on computer support for collaborative learning (CSCL) [3,4]. This conference, which occurs every two years, is devoted to the exploration of the use of technology in collaborative instruction and learning, and is attended by educators, researchers, designers, and students from various countries specializing in a wide array of disciplines. Previous topics discussed at the conference have included designs for tools (microworlds, multi-user simulations), theoretical perspectives (pedagogical theories, constructionism), and empirical methods (assessing student achievement) that promote collaborative learning.

There are several existing software tools that help to facilitate collaborative learning in educational institutions. Although much fewer in number than their corporate counterparts, these tools usually require a smooth integration into the activities they are intended to support in order to be used effectively. Like their corporate counterparts, however, most of these tools also rely on the World Wide Web. Two good examples of tools for collaborative learning are JavaCAP [19] and Belvedere [1,20]. JavaCAP allows
students to author cases in teams using existing cases as resources, while Belvedere is used to analyze scientific problems using inquiry skills. Both tools are discussed in detail in Chapter 2.

While tools such as JavaCAP and Belvedere offer a reasonable degree of collaborative features, they do not provide a lot of flexibility in terms of supporting various kinds of collaborative activities. Some activities are specifically designed to facilitate the generally important goals of sharing and discussing of ideas in a learning community, and the number of tools currently available that can make these goals achievable is relatively few in number. This need for effective collaboration-centric tools has led to research on methods to not only consolidate and revitalize existing educational communities, but also to help create new communities and perspectives on learning in the digital age. The research presented here is focused on constructionist learning communities in particular [7].

Over the past two decades, constructionist learning researchers have provided substantial evidence for their central claim that children learn most effectively when they are engaged in designing something that the care about deeply. To this end, they have created many construction kits that enable children to pursue their own interests and styles of learning in many diverse fields – such as robotics and complexity – many of which were previously inaccessible to them. In the constructionist theory of learning, the artifacts created by children play a central role. They allow the learner to externalize his or her ideas in a form that can be discussed and reflected upon. To a constructionist, the social dimension of sharing what you have made and receiving feedback from the experience of making it or from others around you is an integral part of any powerful learning experience. Therefore, it should come as no surprise that constructionist educators, learners, and researchers would value tools that enable them document and share their learning
experiences. Apart from their capacity in building a growing library of experiences and tangible know-how, collaborative tools are also valued deeply by constructionist educators and learners because they can help form a community of practitioners and learners. This extended community of researchers, mathematicians, scientists, educators, teachers, and learners is a notable example of a constructionist community.

The construction kits created to facilitate constructionist learning have enjoyed great commercial success. Millions of children have used the numerous programming and design environments and activities that have been based or have directly grown out of the research of the Epistemology and Learning Group at the MIT Media Laboratory. The Logo [15] and StarLogo [16] programming languages, the LEGO®/Logo [13] project, and the MINDSTORMSTM [10] product from the LEGO Group® are noteworthy examples. An important remaining challenge – and opportunity – facing the constructionist community is to better document and share the practices, learning activities and stories that are growing around the use of these tools. We will present a detailed scenario in Chapter 3 to give the reader a better sense of not only the nature of the activities that we have in mind, but also of Crossroads, which is the collaborative design notebook that we designed to support these activities.

In order for collaborative tools to support constructionist activities, they need to make use of the right hardware infrastructures. This is particularly the case for collaborative software applications, which are usually designed with specific networking requirements. These requirements range from the simple (one-to-one communication protocols) to the complex (many Internet protocols).

Crossroads is an effort towards the development of the constructionist learning communities described thus far. It is a software system that provides collaborative editing
capabilities in a networked environment, and is designed for use in educational institutions, particularly schools, after school centers, and museums. Learning is an integral aspect of the activities carried out in these institutions, and Crossroads makes it possible to document the results of such learning, both individually and collaboratively. Although the system is designed for use in educational settings, it can be used in a setting that involves groups of people documenting their collective work. Using the system requires a computer network with an intranet infrastructure (at the minimum) and Internet identification.

Crossroads comprises of component tools that provide its collaborative capabilities. Its core component, the Editor, allows a user to create and edit team projects containing different kinds of media, e.g., pictures, video, audio, etc. These projects are saved in a team workspace, and are accessible to other members of the team for further editing. The Editor supports project versioning, which allows team members to access previous versions of a particular project. Crossroads also includes the Administrator, an administrative-level program for administrators who coordinate the activities that several users are involved in. With the Administrator, administrators can add users into the system, and assign them into teams. The system includes two other tools, each of which may be used independently but act primarily as helpers to the Editor and the Administrator. One tool is the Resource Folder, which allows users to access ‘media resources’ – media objects that other users have made public. These resources include some textual annotation, which makes them useful in various contexts. Another tool, the Project Browser, is similar in functionality to the Resource Folder. It allows users to view projects that have been published by other users within the community. Both the Resource Folder and the Project Browser have browsing and searching capabilities that make sharing media objects and projects possible, respectively.
In order to perform their functions, these tools need access to a network resource where all shared data, such as user information and project data, is stored. This access is provided by the server tools of the system. These tools maintain connections to a central database that stores all shared, persistent data used by the system. The main server program, the Standard Server, provides general server functionality. It allows client tools to update and retrieve projects, media resources, and other system data. The Editor, Resource Folder, and Project Browser all make use of this server. The Administrator, however, uses an Administration Server in order to carry out its tasks. This server updates information about users and teams, which is also stored in the central database. Although both servers are distinct system components, they share some functionality.

The entire system as described above leverages the network structure of its host community. When completely used within an intranet as recommended, server access is generally quick, and, as a result, the average response time of the client applications will be relatively small. If the network structure of the community changes as a result of moving all server programs to remote sites, the system can still be deployed since each computer configured for it is required to have Internet identification. The modifications to the system to support this change are minimal.

It is important to note that while the tools provided by Crossroads allow users to collaborate on an activity, they are not designed as the primary agents of collaboration. Using them successfully requires a smooth integration with offline activity, e.g., science experiments, field trips, etc. The manner in which the tools are used depends on the context. Within the context of an elementary school, for example, teachers would be the administrators of the system while their students would act as the users. In addition, it
would be the teachers’ responsibility to determine the settings in which the system would be useful, and the activities that the system’s tools use would complement.

This thesis describes Crossroads in detail. Chapter 2 discusses other existing software that facilitates collaborative work and/or learning. Chapter 3 presents a scenario in which Crossroads can be used. Chapter 4 covers the design and implementation of the system. Chapter 5 provides a preliminary evaluation of the system. Chapter 6 describes potential extensions to the system design. Chapter 7 concludes with a reflective take on the Crossroads project thus far.
2 RELATED WORK

In the past decade, many software systems have been developed to enable and support collaborative work and learning – majority for work than learning – and most of them take advantage of the Web as a new medium for communication. With so many systems, one might ask, “Why design another one?” The goal of this chapter is to present a careful study of three collaborative systems – one designed primarily for the workplace and the other two for learning contexts – to make the argument for the necessity of yet another collaborative tool and lay the foundation for the design rational presented in chapter 4. The discussion in this chapter will be organized as relatively detailed case studies of the following software systems: QuickPlace™ by Lotus® Development Corporation, an application for general collaborative work; JavaCAP by EduTech Institute at Georgia Tech, a collaborative case authoring tool designed specifically for use in elementary and middle school children; and Belvedere by the Learning Research and Development Center (LRDC) at the University of Pittsburgh, a collaborative tool for conducting scientific inquiry.

We believe QuickPlace™ is an ideal representative because it is popular in team-based work settings and it boasts several features that are designed to enhance team productivity. In the case of software for collaborative learning, we selected JavaCAP and Belvedere because both tools were designed for use in educational settings, and each provided an interesting approach to supporting collaborative work that we thought was worth reviewing. JavaCAP was built for team authoring tasks, which was similar to the tasks that we wanted Crossroads to support, and Belvedere was designed for a highly specific kind of activity, which made it an invaluable resource while developing Crossroads. In addition, the implementation of Belvedere was based on an architecture that was integral to
supporting real-time team collaboration, which we wanted to account for in Crossroads’ implementation.

Throughout this review, I have added notes (in italics) that highlight the characteristics of these systems that have informed our design decisions for Crossroads and are particularly useful to keep in mind when reading Chapter 4, which describes the design and implementation of Crossroads.

2.1 QuickPlace™
QuickPlace™ is a Web-based software application that facilitates online collaboration within a team. It is designed to increase team productivity by making it more convenient to carry out team duties that were traditionally done offline without a computer or without collaborative software. From another perspective, it supports collaboration by providing a team with a shared online workspace, which can be used as a storage location for documents related to the team’s work and as a channel for team communication in real-time. QuickPlace™ provides each team member with a homepage, suitably referred to as the member’s QuickPlace™, which serves as a top-level interface to the capabilities available to the member. QuickPlace™ includes other useful features, such as a calendar, a task list, and a milestone tracker.

From his or her QuickPlace™, a user can specify access control settings that dictate which users are capable of browsing and publishing the homepage content, as well as inviting new users. Users invited can be designated as a Reader, an Author, a Manager, or an Anonymous Reader, as illustrated in Figure 1 [14]. Readers only have browsing privileges, while Authors can both browse and publish content. Managers have the highest permission; they can invite new members, add folders and rooms, customize the user interface, as well as
browse and publish new material. Anonymous Readers have only read-access to the homepage's content and are not required to provide authentication to use QuickPlace™. Using these capabilities, a user can easily create a team by first designating himself or herself as a Manager and then inviting other users to participate in the collaborative work.

Figure 1. Designating access to documents in QuickPlace™.

As described above, team administration in QuickPlace™ can be highly decentralized. Although this capability makes it a more flexible tool for the general user, it is sometimes not ideal for certain contexts, particularly in educational institutions, as they exist today. For example, in most elementary schools, teachers maintain complete responsibility for dividing students into teams when they assign collaborative work. Although students in certain cases can form informal groups, teachers act as the primary administrators in coordinating most collaborative activities that are integrated into the school's core curriculum. Though we
designed Crossroads with these limitations in mind, we have allowed for its use in much more free and flexible contexts.

Apart from the standard e-mail, voice-mail, and fax support, QuickPlace™ grants Managers the permission to create virtual inner rooms. These inner rooms are used to further restrict access to a particular set of users. Furthermore, only the people who can enter these rooms know of their existence. Figure 2 presents a snapshot of a page allowing the user to create an inner room.

![Creating an inner room in QuickPlace™.](image)

Figure 2. Creating an inner room in QuickPlace™.

Team members can check out a document from the team workspace provided by QuickPlace™, thus preventing other members from modifying the document until it has
been checked back in. The application also maintains a version history for each document, which is updated whenever the document is checked in. After final revisions to a document have been made, a team member can publish the document. Based on how they set up the workspace before the game, team members can have QuickPlace™ automatically notify other team members when a homepage has been edited, or when new pages have been created.

The versioning mechanism used by QuickPlace™ is rather common in many collaborative software tools, though in slightly different forms. This mechanism has clear benefits in ensuring that all document edits are tracked and synchronized. Crossroads employs a closely similar scheme for managing project revisions.

As an authoring application, QuickPlace™ includes support for styled and colored text and images only. It also allows users to create pages using existing documents such as Microsoft Word files or HTML files. Figure 3 captures some of QuickPlace’s authoring capabilities.

![Figure 3. Authoring content in QuickPlace™.](image-url)
A useful feature provided by QuickPlace™ is the ability to work offline. In the absence of an Internet connection, users can edit a local copy of a page (that was automatically downloaded when the connection was still alive). Once connected to the Internet, QuickPlace synchronizes the local content with its online version.

In summary, QuickPlace™ is a very effective tool for team collaboration. It makes it very convenient for members of a team to communicate and share documents; and, it provides features that help facilitate team organization, such as calendars and task lists. However, despite these advantages, QuickPlace™ has its drawbacks, particularly in its authoring capabilities. While it allows users to compose pages with text and pictures, it does not provide support for time-based media such as video. Also, related to this missing functionality, QuickPlace™ lacks a flexible environment for editing documents. In order to create documents, users have to rely on fixed, standard, HTML interfaces such as text areas, which may be inadequate for some authoring needs. In general, QuickPlace™ and other collaborative tools may also be unsuitable for some collaborative contexts simply because the capabilities that they offer are limited. These contexts may require deeper functionality from a collaborative software application than access control of shared documents and tools for instant communication.

2.2 JavaCAP
JavaCAP is a Web-based educational software tool for collaborative case authoring. It allows students to collaboratively author case reports, which are added to a searchable case library. These reports usually suggest approaches to solving a particular problem or provide results found in a study. Students author cases through JavaCAP’s authoring component, which provides the tool’s core functionality. When applied to the Learning-By-Design
(LBD) context, this component serves as an interface for students to reflect on a recent LBD experience and to document the structured summary of this experience. Once written, these experiences are collected in a library of cases, which in turn can serve as learning models for other students who need to document their own experiences.

JavaCAP was developed for use for the LBD curriculum at Georgia Tech's EduTech Institute. One of the primary goals of the LBD is to explore ways of promoting better science learning among middle graders by having them learn science through solving design problems. In LBD activities, students are divided into small groups and are given design problems to study. They are encouraged to brainstorm ideas on how to address a particular problem, research the material needed to understand the problem, and work on their design. Once they have a solution, they are required to test it in order to gain feedback about their design. This testing process is considered crucial to the learning process, since a failure usually indicates that some data has been unaccounted for. In addition to their design, students are also encouraged to come up with alternative solutions, which they compare with the design that they chose.

Note that JavaCAP was designed to support a particular kind of activity to be conducted in the specific context of primary and middle school science classes. Furthermore, it imposes a certain structure on how this activity is coordinated and how the participants, i.e., the students, will learn and reflect on their learning. This bears a significant difference from Crossroads, which is designed for a broader range of learning and design activities in many different contexts (not restricted to schools), and where each of these activities may be structured quite differently than the activities supported by JavaCAP.

The motivation for JavaCAP was to provide case authoring capabilities for students in such a way that students will be able to reflect on their experiences and as a result be able to reflect on and present their experiences in ways that will allow other students to
understand and learn from them. The JavaCAP research team discovered that students where not inclined or able to reflect on their experiences; for example, quotes one of the team members: “our experience shows, as well, that students dislike reflection unless their teachers have introduced it in a way that allows them to see its use and that many teachers are uncomfortable with helping students reflect – it is hard.” (Shabo et al., 97) They argued that a framework was necessary to both students and teachers to promote reflection. They sought to find ways of providing frameworks that would complement the kinds of design activities that the students were doing, and support reflection over various aspects of their activity, e.g., the skills needed, the background material required, their approaches, etc. They also wanted to develop frameworks that would integrate in a natural way the reflection that occurred during the design process with the reflection that occurred after design.

In order to support the first kind of reflection, the JavaCAP research team developed design journals that allowed students to make their thoughts and ideas throughout the problem solving process explicit. However, they found it more challenging to provide the second kind of reflection, which usually occurred after solving a problem. They wanted to make it more of a natural for children to reflect on their activities and more readily see its value. Using their previous experience with case-based reasoning and case-based libraries, the research team decided to apply the concept of authoring cases to the design activities that the students participated in. They had seen from an earlier study with college students that students enjoyed reading each other's cases and learned well through case authoring. JavaCAP was conceived when the researchers decided to try this concept with middle school students.

JavaCAP allows students to document a previous activity as a sequence of scenes. Each scene is a descriptive summary of an aspect of the students' design experience, e.g., the
problems being addressed, possible alternatives, their solution, etc. Students fill a scene, by uploading text and images to the scene and annotating them. Once the scenes have been filled, students can group them collectively as a case, to which they attach suitable keywords in order to make their work resourceful. The case can then be published and kept in a library that is accessible to all students.

There are four scenes in JavaCAP. The first is the "Problem Presentation" scene, shown in Figure 4, where students specify the problem addressed and the objectives in studying the problem. Following this scene is the "Alternatives Selection" Scene, illustrated in Figure 5, where suggested design alternatives are presented, in a tabular format, allowing them to be weighed against the others according to some specified criteria [8]. In the "Solution" scene, the design chosen is described, as well as the results of implementing it and the lessons learned in the process. The final scene in the case is the "It's A Wrap" scene, which includes additional aspects of the project, which the students considered important or useful, e.g., collaboration experiences.

![Figure 4. The “Problem Presentation” Scene in JavaCAP.](image-url)
With the exception of the "Alternatives Selection" scene, each scene is designed as a two-column notebook screen. The left column provides an unstructured area for students to type in their experiences regarding a particular aspect of the design. The column also allows students to attach a scanned photograph with annotation, to help convey their experiences. The right column, on the other hand, is a structured and sequential interface designed to facilitate the reflection process. It provides features for indexing the case and specifying generic textual details about the case. In the "Problem Presentation" scene, for instance, the left column asks students to identify what they considered important upon beginning the design problem, including a picture, if necessary whereas the right column requires a title for the case, the problem specification, and some learning expectations. This two-column user interface model was designed to facilitate collaboration within the group during the authoring process. One group member could contribute a photograph relevant to the problem being addressed, while another member can define the group's statement of the problem.
The user interface for the "Alternatives Selection" scene is a table of textual components in which students provide a list of suggested alternatives, which are weighed against a set of criteria they used in making their design decision. According to the researchers, this scene is usually considered the most challenging for students since they may not have evaluated their alternatives in such a precise manner.

The user interface for JavaCAP is highly structured, perhaps in an attempt to increase uniformity in the presentation of the work carried out by the various teams. While this structure might be perceived as reasonable, it can also be viewed as a restriction on a student's way of expressing what he or she had learned in the design process. Crossroads, unlike JavaCAP, provides an open editing area for the student to play around and present his or her work, without any structural boundaries. Crossroads assumes that structure grows naturally from the nature of the activities and the interactions amongst students and teachers/mentors.

When used in a well-chosen context, JavaCAP can be an invaluable tool for students to document their design experience in ways that will make this experience resourceful. These benefits can be attributed to the tool’s text and graphical annotation capabilities. In addition, its user interface structure allows students to reflect on their design in an approach that is much more guided than those of other collaborative applications. This guidance structure is provided at a price, however, since the target audience for JavaCAP is limited. The tool is also limited in its editing capabilities. While it supports image and text content, JavaCAP does not handle the display and manipulation of time-based media such as audio or video clips. A student may have collected several video clips of his or her design work, which cannot be integrated into JavaCAP because of this missing functionality. Additionally, owing to its Web-based nature, JavaCAP lacks certain editing operations, including undo-redo functionality, cut-copy-and-paste data transfer, and object dragging. All media imported into JavaCAP are constrained to a particular state and location. JavaCAP also has
its drawbacks as a collaborative tool. Its use is highly dependent on the activity in which it is adopted, and, as a result, it will not be useful in other contexts, not very similar to those it targets.

2.3 Belvedere

Belvedere is another system aimed to facilitate online collaboration among K12 students. Bearing the same name as the project that it was developed for, Belvedere includes tools and materials designed to help students learn and improve their skills in both individual and collaborative settings. Students are divided into teams to analyze a relevant, though simplified, scientific problem. Each team uses the tools provided by Belvedere to search through online materials such as articles, references, etc. that are related to this problem, and select a subset of these for study. Using a provided diagramming tool, they try to understand the problem by constructing a model consisting of abstract ideas and relationships among these ideas. They can also extend the model by constructing hypotheses and providing evidence to support these hypotheses. The system provides an online coach that can help guide the students throughout this process, although it remains dormant until it is invoked or has highly important advice to offer. The online system is tightly coupled with offline activity usually performed by a scientist – talking with other scientists, reading articles, attending conferences, going on field trips, conducting experiments in the laboratory and in the field.

Belvedere was developed by a research team at the Learning Research and Development Center (LRDC) at the University of Pittsburgh. Founded in 1963, the LRDC aims to find new approaches to enhancing scientific learning using new technologies. A common perspective adopted by many LRDC projects is that of the student as an active
learner who constructs his or her own knowledge as much as a scientist constructs socially shared knowledge.

The application domain of Belvedere is in learning critical inquiry skills in science, which differs from JavaCAP's primary purpose. Using the tool, students can develop these skills, which are applicable in everyday life as well as in science. These skills include acquainting oneself with a field of study, identifying a problem of interest, making hypotheses or proposing solutions, locating evidence that supports the hypotheses or solutions, making conclusions based on the supporting evidence, making a summary of the inquiry, presenting the problem study to others, and conducting and discussing the inquiry process pursued by other students.

According to the researchers, students can learn inquiry skills better by being more active in learning science and as a result see science as a process, and not just a collection of facts. Additionally, through the collaborative nature of the project, students can build a community where they work together to conduct inquiries whether they are brainstorming or collecting evidence. They will be able to reflect upon their work throughout the process, and understand how a scientific community works through scientific problems. As students conduct inquiry, they become more adept at discriminating scientific information from non-scientific elements and more skillful in analyzing and reporting such information. They learn the necessary tasks of working with others and respecting the opinions of others, and as well as utilizing others as resources. In effect, the Belvedere project aims to combine two important, and quite different elements: the process of analyzing scientific information, and collaboration, both in a learning context.

Like JavaCAP, Belvedere is targeted towards specific kinds of collaborative activities, particularly those that involve conducting critical inquiry. In addition, it designed for use in contexts that require students
to conduct a substantial amount of research in order to develop a certain perspective on a particular concept or idea. In contrast, Crossroads allows for several styles of research, development, and design. Students can use Crossroads to document work stemming from their research, to experiment with their creative ideas, and to brainstorm about and keep a journal of their design ideas. It could be based on any kind of construction, from thinking of new approaches of representing concepts of time and space to building concrete objects like robots and planes.

As mentioned earlier, Belvedere provides tools and materials to facilitate the learning of inquiry skills. These include:

- a collaborative inquiry database that contains information about real, though simplified, scientific topics
- an inquiry diagramming tool that enables students to formulate their thoughts, and construct hypotheses and relationships between hypotheses
- an expert-based coach that serves as a guide to students throughout the inquiry process when necessary, as opposed to a solution repository
- a chatting facility that enables team members to communicate among themselves
- HTML-based materials that consist of offline articles and other supplementary materials carefully selected for a particular activity upon the teacher's discretion

The manner in which the tools and materials are used is up to the supervisor or teacher. A typical scenario involves a class science project that requires students to do research on a particular topic and provide an analysis or a solution to a problem stated in the topic. The teacher would first select and prepare a resource set of articles and references and other materials that would help students conduct their background research. These materials would be HTML-based, making them viewable on any computer with a Web browser. Once
these materials have been reviewed, students would be divided into teams and required to think about the problem they were assigned and provide a solution for it, with an explanation of their reasoning and solution.

The diagramming tool, which is illustrated in Figure 6, can be used for this problem analysis task [2]. It provides a shared workspace for a team to formulate the assigned problem as a diagram. Members can propose hypotheses, represented as shapes in the diagram that will lead to a solution of the problem. These hypotheses can be related onscreen by lines, which have a thickness value to represent the strength of the relationship. By using this extensible diagram, students see the abstract ideas and their relationships in their design work, which helps to provide a guided focus in their discussions. The state of the diagram is communicated to other team members as it is being edited, through a mechanism commonly referred to as WYSIWIS (what-you-see-is-what-i-see). With the included chatting module, team members can communicate and share ideas during this editing process. After the inquiry has been completed, the team can present these findings to the class. Through the presentation, team members can receive feedback about their work from their peers regarding their inquiry approach, the evidence gathered, etc. They could use this feedback to reevaluate their approach and perhaps repeat the inquiry to account for some new evidence that was previously unaccounted for. Also, such feedback may be useful to the team as far as reflecting on their work. The team can also write a report describing their inquiry process and results. This report is published as HTML, and is accessible to students in the particular team and other teams, as well as teachers and parents regardless of physical location. The only necessity is an Internet connection.
Figure 6. The inquiry diagramming tool in Belvedere.

The above scenario illustrates that the functionality provided by Belvedere’s diagramming tool can be used for reasoning about the abstract relationships related to a particular scientific concept. Although the system as a whole provides a sufficient level of collaboration in activities that require this kind of reasoning, Belvedere, unlike Crossroads, is constrained in its usability to contexts that involve team collaboration in other ways.

The system architecture is based on a client-server model. The computers that students, teachers, and parents use contain the necessary client software—Web browser, chat, diagramming tool, and coach—which interacts with the server tools. Since several
updates to a client's state of the system are asynchronous, i.e., chat messages, coach advice, diagram state, each client has a listener module running in a separate thread to listen for these updates from the server. As a result, the server maintains the responsibility of maintaining connections between clients. This mechanism enables a student to pick up a team discussion from where it left off. It also serves as a site of storage in the event of a client shutdown, etc.

The system was built using widely supported computing standards, such as Java, CGI, Netscape, etc. This was done to achieve a high degree of interoperability and reusability, while minimizing the cost of maintaining the system. As a result, new functionality can be easily integrated into the existing system, and certain aspects of the system may be reused in other software designs, possibly related to educational collaborative software.

Belvedere uses several technologies and computational paradigms to achieve team collaboration, which can be suitably adopted into the design and implementation of other collaborative software projects including Crossroads. Of particular interest is the overall separation of Belvedere's functionality into client and server components, and how the server is used to keep track of client connections to enable collaborative functions such as chatting.

Belvedere provides several key features such as chatting that make it an effective collaborative tool for the kind of the activities that it was designed to support. Furthermore, it benefits from a flexible and modularized software design, making it possible to extend the system to support new features. However, Belvedere is not without its drawbacks. Like JavaCAP, its audience is limited to only those conducting collaborative inquiry. While this makes its usability focused, it also constrains the overall impact of its collaborative features within the educational community. In addition, the system does not allow for users in a
team to regularly share parts of their ongoing work with other students not in the team. The only form of team-team sharing that it supports occurs through project presentations and online reports shown and distributed in class. As a result, the collaborative strength of the system within a decentralized community may be limited from reaching its full potential.

2.4 Closing Remarks
As observed from the study of the existing software systems described in this chapter, some tools support collaborative learning activities in limited contexts. Some tools, while designed for collaboration, do not integrate capabilities to promote any form of learning. In addition, tools that allow the learner to document the planning, process, and results of collaborative activities are few in number, and do not provide support for manipulating rich media content such as audio and video. Crossroads, the system described in this paper, serves to not only support the process of collaborative learning in a wide variety of activities, but to also capture the process as well. The following chapter describes a scenario that highlights how Crossroads can be used to accomplish both tasks.
3 SCENARIOS

Crossroads is designed to enable and support documentation and discussion around constructionist collaborative learning activities, in a variety of contexts with many different network requirements. In order to get a sense of the kind of scenarios Crossroads would be ideal for, let us consider one hypothetical scenario in detail. The scenario, though hypothetical, is based on the experiences of researchers in the Epistemology and Learning Group at the MIT Media Laboratory in a variety of settings — workshops, after school centers, and classrooms — with incorporating the constructionist tools that they have developed in workshops over the past decade. It is presented as a narrative describing a fictional school that adopts Crossroads into its learning curriculum. The source of this narrative is mostly imaginative, though inspired by prior observations of learning contexts such as workshops either directly or indirectly (through video). It is important to note that this narrative includes both implemented and planned features of Crossroads.

Ms. Wilson is an eighth-grade schoolteacher at the Children’s Design School (CDS). A small middle school in Boston, Massachusetts, CDS has been able to obtain constant Internet connectivity with the help of several sponsors. It has a local area network (LAN) infrastructure, through which all computers in the school access the Internet. In addition, there are two computer laboratories in the school, and each having ten computers. Each of these computers contains various software packages, including software for converting recorded video data to electronic files and for audio recording. The entire computer and network infrastructure of the school is managed by a small team of network administrators, who regularly visit the school to ensure the network and other hardware components are functioning properly. These administrators also ensure that all software, including
Crossroads, has been configured successfully on the network. Specifically, they have
installed the Crossroads server programs in the several high-memory, high-performance
server machines owned by the school, configured all the computers in the laboratories to use
the Crossroads client software, and performed comprehensive tests to ensure that the client-
server communication line was working well. With this architecture, should the school’s
Internet connectivity become scarce, no major configuration changes need to be made to
ensure that Crossroads is still functioning, as long as the school’s intranet is still active. Also,
for increased availability, additional servers can be installed within the intranet.
Furthermore, in the presence of reliable Internet access, more servers can be placed in
locations outside the intranet, and these can be configured for CDS in particular.

Everyday, Ms. Wilson teaches her students, twenty in number in each class, subjects
ranging from Math to Science to Engineering. Some of the subjects she teaches require her
students to use the computer laboratories to do an assignment, e.g., writing a report,
conducting research on the Internet, etc. On one particular day in the month of February,
Ms. Wilson, spirited in the constructionist/constructivist kinds of learning, decides to
brainstorm activities that will allow her students to work together on projects, and learn
from each other in the process.

She decides on the following activity, which she had coordinated the previous
semester. The activity is to build doll-like models of creatures. Particularly, the activity
would require the building of the creatures’ limbs such as hands and legs from a provided set
of materials. Several heads and chests will be provided, although each student participating
in the activity could build his or her own – though only as extra credit work. The aim of the
activity is to give students a more concrete appreciation of and to generate discussions
around the curricular material on the salient differences in the designs and functional
attributes of human and animal limbs. Ms. Wilson divides each of her classes evenly into five teams of four students per team. The limb design and creation part of the activity is expected to last eight weeks. At the end of this period, each team would have to prepare a public presentation and a final report that documents their design process as well as the final artifacts.

Ms. Wilson has decided to use Crossroads as an integral part of this activity. Specifically, she wants her students to use it as a tool for authoring and sharing. In order to make this possible, she makes sure that her students are registered Crossroads users and creates Crossroads teams for the teams that they belong to. Since she has never setup Crossroads for her class before, she runs the Crossroads Administrator tool to register her students into the system. She assigns each of her students a username/password pair, which each student was required to specify at an earlier time. Once the students have been entered into the system, a virtual activity is created, named “Body Building.” Virtual teams are created and added to this activity through the Administrator, and students are assigned to their respective teams. Figure 7 captures this process. It is time to start the activity.
Day 1: Ms. Wilson introduces the activity to her students. She provides them with a set of materials of several kinds, both traditional (wood, metal, plastic, paper) and computational (LEGO® MINDSTORMSTM products), and points out that they are free to create any sets of legs that they can think of. She also obtains passes for them to use the
computer laboratories, in case they need to do research on building limbs. As a useful but optional research guide for her students, she mentions a website that references existing projects that were created by students in her former class who participated in this activity the semester before. She also notifies them that they have been registered as Crossroads users and that they should use Crossroads to document their progress.

**Day 2:** Team 2, which consists of Harry, Helen, Leann, and Leroy meet to decide how to divide up the work. Harry and Helen choose the build various kinds of hands and arms, while Leann and Leroy decide to create legs. They agree to spend the rest of the day sampling the available materials and start building the project the next day.

**Day 3:** Each team is already getting involved in the activity. In Team 2 particularly, Helen and Leann have been researching websites for tips on how to build hands and legs respectively and have gained some input on doing so. Harry and Leroy, however, decided to forgo research and start building limbs on their own. They were able to come up with some sample limbs by the end of the day. They borrowed a video camera from the school's instrument store, which they used to take some pictures of their work, as well as some short video clips.

**Day 5:** Harry and Leroy convert the pictures and video they have collected to media files, which they import into empty projects using the Crossroads Editor, as illustrated in Figure 8. Helen and Leann, on the other hand, are still researching on limbs and are expecting to gather enough hints for them to easily do their work.
Below are samples of media objects that I have been collecting over the past few days.

Included are two pictures of animals and a short video clip showing the process of lobster fishing. Although these are not quite relevant for our project, I wanted to play around with Crossroads Editor before I started doing real work!

**Figure 8.** Leroy using the Crossroads Editor to create sample projects.

**End of Week 1:** Helen and Leann catch up with the work that Harry and Leroy have been doing, respectively. Helen views all the pictures and video clips of hands that Harry has been collecting and places them in the Crossroads Resource Folder, making them available to other teams. Also using the Folder, she comments a little on the projects that
Harry had created, describing the various pictures and video clips that he had added earlier in the week. Leann decides that she should publish the most recent version of the project that Leroy has been creating, even though there is still much work to be done. She does this using the Editor, and asks Helen to check if she can view the newly published project in the Crossroads Project Browser. Helen confirms. “Every other team can now see what we’ve done,” says Leann excitedly. Each team also becomes aware that Ms. Wilson has been reviewing their work thus far, owing to the comments she sent to the team via the Administrator regarding their work, as well as the feedback she gave to each individual student. Harry tells the other Team 2 members that Ms. Wilson said that his pictures vividly captured a wide variety of hand forms. “My pictures do make a difference!” he tells Helen.

End of Week 4: Team 2 has built a wide variety of limbs that are suitable for most kinds of motion – walking, crawling, jogging, running, etc. With these limbs, they have created a few simple bodies, some completely robotic, that can perform some interesting programmable motions such as dancing from side to side and raising a hand up and down. Leroy decides at this time that they should create a project that includes the team’s combined efforts. The remaining team members agree. Leroy then creates a project titled “Hand in Hand, Leg in Leg” in which he places images, text, and video clips describing the legs that he and Leann have built so far. He also edits some of the text to make it more apparent that the content he just added is part of a document with a wider focus. He encourages both Harry and Helen to add their content to this project he just created, and Helen mentions that she will do that tomorrow. Helen says that she will publish the version of the project that she creates once she is done. Also, Harry lets everyone in the team know that, in addition to his construction work, he has been documenting the team’s plans and progress using the Editor. He mentions that twice a week from home, he reviews all the versions of the
projects that the team has created since the activity began using the Editor's version browsing capabilities, and has been summarizing his review results in a special project called “The Distance So Far,” which is open for editing by other team members. He also points out at this time that, out of interest, he has been annotating the various media objects (pictures, audio, video) used in the projects that had not been annotated yet. He clarifies that for some of the objects previously did not have any annotation, and, as a result, he came up with some text to describe the object in the context of the containing project, and suggests that the team member who added the media object should review the annotation to ensure that it captures the expected idea. Meanwhile, Ms. Wilson uses the Project Browser (for the first time in a week) to view the progress each team has made, and is impressed with the results she sees.

**End of Week 8:** Team 2 has been able to create a well-composed project report of their limb-building approach that they can present. According to Leroy, the report has about 30 versions, starting from the beginning of the month. The team decides to make last minute edits to the report, thereby creating more versions for the system to keep track of.

**End of Week 9:** The teams present their works to the class using a projector that displays the current view of the Project Browser. Each team discusses their strategy for building limbs, what their limbs can do, and what they found most challenging and rewarding about the process. Finally, they entertain questions from the class audience.

The above scenario is one example of the various scenarios that Crossroads can be used for. Although mostly imaginative, the scenario does illustrate some of the major capabilities that Crossroads offers – editing, viewing, and sharing project content, annotating media objects, accessing different versions of a project, integrating rich multimedia into
projects, administering users into teams, giving teams and students written feedback on their work, etc. It also serves to provide a sense of how Crossroads could be integrated into a constructionist/constructivist activity, although there are many ways to support this integration based on the coordinator of the activity.
4 DEVELOPMENT

4.1 EVOLUTION
The design of Crossroads originated from the development of an idea of a similar, though less functional, system. This system comprised of an editing tool for creating multimedia projects, an administrative-level program, and a remote server program with local access to a central database. The system would be deployed in elementary schools where teachers have access to a handful of computer laptops but Internet access was usually scarce. Using the administrative program, teachers would setup the editing tool on their laptops so their students could use it to create multimedia projects. All administrative work done by the teacher would require Internet access. The editing tool, however, would be used in the absence of a network connection, and all documents created with it would be saved on the local machine. At the end of the day when network connectivity was possible, teachers would run the administrative program on each laptop to synchronize the projects saved on the laptop with those located on a particular remote server. Once started, this synchronization process required little attention from the teacher.

The idea described above was further developed, resulting in Crossroads. Most importantly, this idea was extended to support collaboration. The intent in supporting this extension was to not only allow people to collaborate on a task, but to also encourage the sharing of ideas while performing this task. Such sharing can lead to enhanced learning, where the participants involved would learn by doing and also learn from each other. This collaborative aspect would apply to the whole community; for example, an entire school, or parts of the school where the system would be in use. The support for such collaboration
brought a few requirements, most of them network-related. One such requirement that was
inherited from the original idea was to allow local storage of edited content in the absence of
an Internet connection. This appeared to be a complex requirement in the goal for
achieving collaboration at anytime from any appropriate location within the collaborative
community. A middle ground was achieved by requiring that system be deployed within an
intranet infrastructure. Although an intranet was not a strict requirement, it (as opposed to
the Internet) was the recommended network infrastructure for reasons of isolated
communication and improved performance. The client and server tools would be installed
on computers located within this network. All data would be saved instantaneously on the
server’s database, which was also located within the intranet, without requiring any
synchronization.

As the collaborative system it was imagined as, Crossroads was designed to be used
completely as a standalone system with network access. The only other viable environment
it could be run in was on the Web. That possibility was rejected for several reasons. An
important reason was that, recently, only a few collaborative applications were designed as
standalone programs; the majority of them ran on the Web. Furthermore, the number of
education-based collaborative applications within the domain of standalone collaborative
tools was relatively few. Also, while Web-based collaborative systems had the major
advantage of allowing people to collaborate and communicate from any location at any time,
they sometimes provided an inadequate sense of interaction. This limitation can be
attributed in most part to the stateless HTTP protocol on which the Web activity is based
on. Although several collaborative systems on the Web have incorporated a rich state model
into their functionality, they sometimes require that client machines meet certain system
requirements or have certain software modules installed on them, which may not be feasible.
A closely related issue is that collaborating via the Web sometimes suffers from a large degree of decentralization and incompatibility. A user editing a document on one machine may require that other users working on the same document need to have the same editing software installed on whatever machines they may be working on, which might be impossible, given the client systems. Even if users have the same editing software, they may be outdated in several clients. This creates portability issues. In addition, some Web-based collaborative editing software tools include support for mainly pictures and text. Only a few support time-based media, such as audio and video media. Even purely Web-based collaborative tools that include features such as notifying certain users when state changes occur, e.g., a user modifies and saves a team project, may suffer from a lack of instant interactivity, since they usually rely on sending emails to the other members of the team. These properties of Web collaboration suggest the need for highly interactive, collaborative software systems that are easy to install, provide rich multimedia editing features, and ensure data portability across client machines with different system features.

4.2 DESIGN
The design of the Crossroads system comprises of the individual designs of the system's client and server components, and how these components interact. Each of these components is designed to be extensible, reusable, efficient, and easily modifiable.

The design of each tool is described through an overview of the tool's functionality and the software abstractions responsible for this functionality. A module dependency diagram is also provided to illustrate the interactions among these abstractions. The notation used in the diagram is based on an object modeling notation used by Barbara Liskov and John Guttag, both world-renowned experts on programming methodology [11].
Throughout the design, it is important to note that Crossroads' functionality is described within the context of a site in which the system has been installed. As a result, all references to Crossroads' tools, resources, users, and administrators used in the design apply to a particular installation site.

All abstractions referenced in the design can be found in the appendices.

4.2.1 Architecture
The overall architecture of Crossroads resembles a three-tier model of a client, a server, and a database. The client side consists of all the client applications, i.e., the Editor, Administrator, Resource Folder, and Project Browser. The Standard Server and Administration Server play the server role. The database role is fulfilled by the database accessed by both servers. This architecture is adequate for the needs of the system, since mechanisms for networking and centralized storage are key to the system's goals.

The Crossroads architecture as described is shown in Figure 9. Although other network infrastructures are possible, e.g., Internet-based infrastructures, the recommended infrastructure for Crossroads is an intranet, for reasons of speed and fault isolation. Many Standard and Administration Servers can be installed and run within the host infrastructure, but all most share the same database.
4.2.2 Client Applications
The client tools in Crossroads comprise of the Editor, the Administrator, the Resource Folder, and the Project Browser. Each of these tools is a graphical user interface application that maintains a network connection to a Crossroads server.
4.2.2.1 Editor
The Editor allows a user to create and edit team projects. It provides most of the functionality performed by a general-purpose document editor. However, the Editor extends this functionality to provide capabilities for adding audio and video content to the currently open project. A user can start an Editor session by supplying a username/password pair, which is used to determine the teams that the user belongs to and, in turn, the projects the user has access to. This operation and many others performed by the Editor rely on a connection to a Standard Server. The Editor uses this server to store and retrieve project data, among other things.

Projects created and saved by the user currently in session are accessible only to other team members.¹ When a user opens a project through the Editor, he or she locks the project; other team members attempting to open a locked project can only do so in a read-only mode. A locked project becomes unlocked when the user who first opened the project closed it, whether explicitly or not. A project is closed automatically when the user creates a new project, opens an existing project, logs out, or quits the application. When a user edits a project using the Editor, he or she is actually editing a version of the project. The version history of a project is updated when the user saves the version currently being edited. The user can also designate a particular version of the current project as published. Once published, all Crossroads users will be able to view that version of the project via the Project Browser.

The heart of the Editor functionality lies in its editing capabilities. Using an editing panel provided by the Editor as a placeholder for media content associated with the current project, the user can perform standard editing operations such as cut, copy, and paste. In

¹ Projects that have also been marked as published are accessible to the user community.
addition, the Editor provides capabilities for undo/redo actions and media-specific editing, e.g., style changes to a text selection.

It must be noted that the Editor provides no strict content structure. In other words, the visual context in which the media objects exist does not specify any spatial constraints on the media objects. In other words, all media objects in the context are in 'free space.' An object can be dragged to any location, which sometimes results in overlapping between objects. An alternative approach might impose a strict layout scheme on all objects in the media panel. Using such an approach could offer some benefits over the current approach. For example, it may provide the feel of document structure that is intuitive to many.

The CrossroadsEditor Abstraction

The CrossroadsEditor is the target abstraction of the Editor application. It maintains the Editor's state (current session, team, project, and version) and provides an interface for updating this state as the result of an operation, e.g., creating a new project. In addition, the CrossroadsEditor maintains a media panel, modeled by the MediaPanel abstraction, which manages the media contents of the current version of the current project. Depending on the editable state of the project, the media panel may or may not allow the user to edit the contents.

In addition to maintaining the current state of the Editor, the CrossroadsEditor uses several EditorManagers to carry out user operations supported by the Editor. There are currently four different types of EditorManagers – FileManager, EditManager, MediaManager, and ToolManager. A FileManager is responsible for performing project-level operations. These operations include creating, opening, closing, saving, publishing, and
browsing projects. The EditManager abstraction provides standard operations, e.g., undo, redo, cut, copy, paste, clear, etc., for editing onscreen content. MediaManagers import media sources from the file system into the media panel and provide media-specific functionality such as editing text media and setting the start frames of time-based media. The ToolsManager provides access to a set of useful tools used by the Editor, which includes the Resource Folder and the Project Browser.

The network communication between the CrossroadsEditor and a Standard Server is two-way to support the asynchronous communication framework needed for capabilities such as chat. In one direction, the CrossroadsEditor uses a ServerConnection to access the server. Once this connection has been established, the server also gains access to the machine that the Editor is running on through a ClientConnection object maintained by the Editor.

The CrossroadsEditor abstraction also satisfies the user interface requirements needed by the Editor. In addition to building standard display components such as buttons and labels, the CrossroadsEditor provides functionality for prompting the user for information when needed. It also uses its MediaPanel to display the media contents of the Editor's current project and version. The editable status of MediaPanel can be toggled on and off throughout an Editor session, depending on whether or not the current project was locked.

The module dependency diagram shown in Figure 10 illustrates the abstractions that the CrossroadsEditor depends on.
Figure 10. The module dependency diagram for the Crossroads Editor.
4.2.2.2 Administrator

The Administrator is an administrative-level tool for managing users and teams. Like the Editor, the Administrator makes use of a session that identifies the administrator is currently using it. Each administrator maintains a mutable list of users, and he or she can assign users on this list to a particular team. The Administrator also maintains a connection to an Administration Server, which it uses to record activity, team, and user updates.

Although the Administrator and the Editor are used for different purposes, both tools have a few properties in common. Furthermore, in certain cases, the Administrator can be viewed as the administrative counterpart to the Editor. Like the Editor, it is a session-based top-level tool, which can update to a Crossroads server in significant ways.

Conflicts can arise if an administrator is permitted to make certain changes affecting a user via the Administrator while the user is simultaneously logged in to the Editor. For example, allowing the administrator to remove a user while the user is editing a project would threaten state consistency. Owing to this conflict and other related ones, the Administrator prevents the modification of user, team, or activity information while a running Editor instance is actively using that information at the same time. Thus, in our example of an administrator removing a user from the system, such an operation will be disallowed if the user is logged in to the Editor at the same time.

All server updates carried out by the Administrator are instantaneous. In other words, the administrator is not given the option to commit a sequence of changes to the server atomically, as a transactional database would allow. This approach maintains an adequate degree of consistency across entire system, since the administrator's view of the users, activities, and teams always reflects the server's view before and after each
administrative operation. While a transactional approach offers the advantage of atomicity over the instantaneous update approach, it suffers from considerable increases in memory complexity, and was thus discarded.

The Administrator has the distinction of being the first Crossroads client application first executed on a newly installed Crossroads system. As a result, it can be thought of as the user configuration engine for the entire system. Once users and teams have been added to the system through the application, users can create team projects via the Editor. The Editor also allows users to publish their projects and make the media sources used in these projects shareable, which in turn, makes the Project Browser and Resource Folder tools usable.

The core functionality of the Administrator lies in its configuration capabilities. It provides the administrator with a high degree of flexibility in setting up activities, teams, and users. For example, an administrator can place a user in multiple teams and can use various naming formats for a certain attribute (username, activity name, team name, etc.). The only major requirement on the latter is that each name specified is unique in the system for the attribute that it represents. For example, only one user can own the username ‘jsmith’. An activity could also be named ‘jsmith’, as long as no other activities have that name.

Like users, administrators also need to be added to the system database before they can configure their workspace. Currently, system administrators at the MIT Media Laboratory perform this operation manually, which can be sometimes a burden for Crossroads administrators. An approach to make this task more convenient is to automate it by having the administrator to specify a username and password pair using a site at a remote Web server. Once this information has been stored remotely on the system administrator’s end, additional work needs to be done to ensure that the information is relayed to the central
database situated at the administrator’s site. This operation is planned for a future system design.

The CrossroadsAdministrator Abstraction

The CrossroadsAdministrator serves as the target abstraction for the Crossroads Administrator application. Its dependency structure bears a striking resemblance to that of the CrossroadsEditor. The state maintained by the CrossroadsAdministrator consists of primarily the current session and administrator. Since each administrator is associated with a set of users, activities, and teams, these may also be explicitly maintained as additional state. However, for memory efficiency, it is recommended that this state be kept until it is needed.

Like the CrossroadsEditor abstraction, the CrossroadsAdministrator keeps a set of function managers – AdministratorManagers – that carry out most of the administrative operations. FileManagers and ToolsManagers are the currently supported AdministratorManagers. A FileManager provides an interface to most of the operations specific to the Administrator, such as setting up users, activities, and teams. The ToolsManager offers a set of useful tools for the Administrator. Although these tools resemble those provided by the CrossroadsEditor’s ToolManager abstraction, the CrossroadsAdministrator’s ToolManager may be extended to include administrative-only tools.

The CrossroadsAdministrator also maintains a network connection to an Administration Server, which is represented by the AdminServerConnection interface. Using this connection, the Administrator can perform administrative-level operations on the server side, such as adding and removing teams. Since the Administration Server and Standard Server share some common functionality, this connection can also be used to carry
out several operations provided by the Standard Server, e.g., retrieving the list of users in a specified team. The Crossroads Administrator also provides a ClientConnection object, which is used by the Administration Server that it is connected to.

The abstractions used by the Crossroads Administrator are shown in Figure 11.

Figure 11. The module dependency diagram for the Crossroads Administrator.
4.2.2.3 Resource Folder

The Resource Folder allows users and administrators to view currently available media resources. A media resource denotes a media source tagged with reusable bits of information that has been made public by a user, who is also referred to as the creator of the resource. The tagged information includes keywords provided by the creator, which serve as primary annotation to the media source, and optional comments provided by any system user regarding the resource, which serve as secondary annotation. These annotation elements may range from an idea about the contexts in which the media source is used to a list of words associated with the contents of the media object.

Media objects are added to the folder via the Editor as media resources. Currently, pictures, audio and video clips are the only kinds of media that can become resources, although a future design will allow for other kinds of media, including text documents, HTML files, and Microsoft® Word documents. A resource is saved remotely and added to the Standard Server using the Editor's network connection. The current session user of the Editor becomes the creator of each resource created during a session. The user is required to enter keywords for the resource before it can be saved. He or she can modify these keywords at a later time, also using the Editor. Comments can be added to a particular resource using the Resource Folder. Since the Resource Folder does not require a session to operate, these comments are anonymous. There is currently no limit on the size of the comments, though a future design will most likely specify an upper bound on this size.

The Resource Folder uses a network connection to retrieve the list of available resources. If the Folder was launched from a top-level Crossroads application such as
Editor or the Administrator, it uses the network connection of that application instead. Otherwise it obtains a connection to a Standard Server from the server itself.

In order to view a media resource, a user can use the browsing or searching functionality of Resource Folder. Browsing support is provided through an appropriate interface that allows the user to view the available media resources categorized by type. The user can also search for media resources using a specified set of parameters. This set of parameters currently includes the media resource types to search through, and a keyword query for matching. The parameter set may be extended to include other properties such as the resource creator in a future design. The search is conducted on the Standard Server that the application is connected to, and the results, if any, are returned to the folder, and are listed. Each media resource in this result list can be then selected for viewing.

All media viewing occurs through an onscreen panel provided by the Resource Folder. The Folder maintains a current media resource, which is shown in the panel. At any point in time, the current media resource is the most recently selected media resource, in either the browsing or searching interface. If another media resource is selected, the panel's contents are replaced with the selected media resource. Also, the media resource currently shown on the panel can be selected by the user and transferred to the system clipboard. This mechanism is useful when the user wants to transfer the media contents of the resource to a running Editor.

The Resource Folder employs a notion of refreshing to allow view the most recent listing of media resources available. Currently, this capability is manually operated, and, as a result, the user has to explicitly refresh the Folder to view the most recent information. A clear alternative is to automate this capability, so that it occurs at statically (or dynamically) defined intervals or when a new resource is created. This approach has the advantage
maintaining consistent views of the resource database among all clients and servers. However, it can also cause interruptions during the use of the Resource Folder, which some users might find undesirable, as was thus discarded.

The CrossroadsResourceFolder Abstraction

The CrossroadsResourceFolder abstraction models the structure of the Resource Folder tool. Its state comprises mainly of the currently chosen resource in the application. Owing to this small amount of information, the CrossroadsResourceFolder provides relatively less operations than the Editor and the Administrator.

The CrossroadsResourceFolder maintains a connection to a Standard Server. It primarily uses this connection to retrieve the media resources available in the system. Unlike the CrossroadsEditor and CrossroadsAdministrator, the CrossroadsResourceFolder does not maintain a ClientConnection. The reasoning for this is that the Resource Folder is intended to be used as a tool that merely retrieves specific information from a server and presents this information to the user. It sometimes updates the server, though in a minimal way, e.g., the addition of comments to an existing media resource. The Resource Folder can therefore be viewed, for the most part, as a simple presentation tool. It was not designed to allow the server to communicate with it or the machine it is running on. Although this communication capability did not seem to impose major constraints on the application's functionality, it also did not appear to prove useful.

The CrossroadsResourceFolder also consists of operations for enabling resource browsing and searching. It supports browsing through the resources of a particular media type and searching through various combinations of media types. The searching mechanism relies on the keywords and comments of existing media resources for string matching.
All media resources selected in a Resource Folder are viewed using an uneditable MediaPanel object maintained by CrossroadsResourceFolder. Although uneditable, the media contents maintained by the MediaPanel, i.e., the media object of a selected resource, can be transferred to the system clipboard through a 'copy' operation provided by CrossroadsResourceFolder. Once in the system clipboard, the media object can be pasted into the media panel of a running Editor instance.

The module dependency diagram shown in Figure 12 illustrates the abstractions that compose the CrossroadsResourceFolder.

![Figure 12](image)

**Figure 12.** The module dependency diagram for the Crossroads Resource Folder.

### 4.2.2.4 Project Browser
The Project Browser allows users to browse projects that have been marked as published by other users in the system via the Editor. A project is marked as published by selecting exactly one of its versions to be the published version. Thus, it is the contents of this
version that are viewed in the Project Browser whenever the project is selected. The published version might change, since a user authorized to edit the project may designate another version of the project as published. The user currently viewing the previous published version can view the new published version by ‘refreshing’ the Project Browser, which works in a manner similar to the Resource Folder. For this reason, the Project Browser can be thought of as the project-level version of the Resource Folder. It maintains a network connection, which it primarily uses to retrieve all publishable projects. This connection may be obtained from the Crossroads top-level application that it was started from, or obtained directly from a Standard Server. The Project Browser also provides user interfaces that allow a user to browse and search through publishable projects. In addition, it maintains a media panel to hold the contents of the currently selected project.

Although the Project Browser bears several similarities to the Resource Folder in terms of general functionality, it does not offer some capabilities offered by the Resource Folder. For example, it does not support the addition of anonymous comments to a particular project. The rationale behind this lack of support is that comments may not be as useful for projects as they can be for media resources, as the publishable version of a project frequently conveys enough textual media within it to be resourceful. Nevertheless, a future version of the system may incorporate them into the Resource Folder in ways that might be resourceful to others. Another Resource Folder feature absent from the Project Browser is the mechanism to transfer specific sections of the currently published project unto the system clipboard. This feature was not incorporated into the Project Browser because some of the media object types that are supported may not be resourceful in other contexts, e.g., text media. Also, by publishing a project, a user may have not intended for the project’s
media objects to be downloadable as well. In the event that the user did intend to make these objects downloadable, he or she can use the Resource Folder for that purpose.

The CrossroadsProjectBrowser abstraction

The CrossroadsProjectBrowser abstraction is the target abstraction of the Project Browser application. Its state consists primarily the currently selected project. It uses the ServerConnection abstraction to retrieve publishable projects from the Standard Server.

Like the CrossroadsResourceFolder abstraction, the CrossroadsProjectBrowser provides capabilities for searching and browsing content, notably published projects. Currently, all projects can be browsed through by first selecting the activity it belongs to. Once an activity has been determined, the projects within that activity can be browsed through using the project’s title as a unique identifier. A future design of the Project Browser might apply other browsing policies. Searching for published projects is carried out using the ServerConnection interface, and is limited to title searches. Similarly, this capability can be extended later to support other forms of searching.

The media panel used by the Project Browser is modeled by the MediaPanel abstraction. This abstraction is also used by the CrossroadsEditor and CrossroadsResourceFolder abstractions, and thus increases the software reuse in the system. This panel’s contents are uneditable and currently unselectable; its contents can only be viewed.

The module dependency diagram shown in Figure 13 shows the abstractions used in designing the Project Browser.
Figure 13. The module dependency diagram for the Crossroads Project Browser.
4.2.3 Server Applications

Crossroads offers two server applications, the Standard Server and the Administration Server, for managing the data used by entire system. Both servers share a common set of operations. In addition, each server maintains a connection to same database, usually located within the intranet hosting the system. This database acts as storage for all information about users, activities, projects, teams, media resources, etc., maintained by the system.

4.2.3.1 Standard Server

The Standard Server acts as the gateway to all shared, persistent data used by all non-administrative client applications. Its primary purpose is to help these applications perform their tasks by providing an interface for retrieving specific parts of the current system data, as well as updating this data.

The Standard Server also performs some operations not directly requested by client applications, such as session management and project locking. Applications can use the interface provided by the server to check the state of these operations. For example, using this interface, an Editor can determine whether a particular project is locked, and who locked it. In addition, the server also maintains a list of all users currently in an Editor session, and a connection to each user's machine. This information can be used to track system use, as well as support additional features such as chat and other user-to-user interaction capabilities.
The CrossroadsStandardServer Abstraction

The CrossroadsStandardServer abstraction is a procedural abstraction that starts a Standard Server on an anonymous port. It is responsible for creating a ServerConnection object, which acts as the central point of reference for all client applications that need to access the server.

The ServerConnection abstraction also maintains a connection to the central database via a DatabaseAccess object. Since a significant amount of the operations specified by ServerConnection require retrieving and updating the system's shared data, ServerConnection significantly depends on its DatabaseAccess for its operations.

In order to allow communication with a given client, a ServerConnection also uses the ClientConnection abstraction. This abstraction comprises of operations that the ServerConnection can run on the client side. Such operations include downloading media source data from the client. Also, the ServerConnection can provide client-to-client communication using the ClientConnection abstraction, making it possible to support chatting functionality.

The CrossroadsStandardServer abstraction graph is shown in Figure 14.

![Figure 14](image)

**Figure 14.** The module dependency diagram for the Crossroads Standard Server.
4.2.3.2 **Administration Server**

The Administration Server can be thought of as the administrative counterpart to the Standard Server, much like the Administrator can be thought of as the administrative counterpart to the Editor. The server is designed to support the functionality provided by the Administrator program. Using a connection to the server, an Administrator can update information about the users, activities, and teams associated with the administrator currently in session. The server also provides access to several operations also offered by the Standard Server, such as retrieving the members in a particular team and obtaining a list of available media resources, for example.

**The CrossroadsAdministrationServer Abstraction**

The CrossroadsAdministrationServer abstraction is the target abstraction for the Crossroads Administration Server. It can be viewed as the administrative counterpart to the CrossroadsStandardServer abstraction, even though it actually extends the functionality provided by a CrossroadsStandardServer. It maintains an AdminServerConnection, which extends the ServerConnection abstraction, and an AdminDatabaseAccess object, which is an extension to the DatabaseAccess abstraction. These extensions are primarily provided to offer administrative-level functionality to client application abstractions while promoting software reuse.

The module dependency diagram shown in Figure 15 illustrates the abstractions that the CrossroadsAdministrationServer depends on.
4.3 IMPLEMENTATION

4.3.1 Client Applications

4.3.1.1 Editor
The implementation of the Editor is divided into five main modules. These modules implement functionality related to session management, project management, general editing, media manipulation, and media panel editing respectively, and are described below as such. Throughout this description, it is important to note the following:

- All operations except the login and quit operations require that a current session and user exist.
- All server operations are executed on the Standard Server to which the Editor is connected.

Figure 15. The module dependency diagram for the Crossroads Administration Server.
• When the current project needs to be saved as the side-effect of another operation, e.g., logout, the user can choose to a) save the project; b) not save the project; or c) cancel the save request. If the user chose the either of the first two options, its corresponding operation is carried out normally. In this case, the project is closed if the calling operation changes the current project. If the user chose c), the save request is canceled and the calling operation is aborted. Once a project is closed, any lock on the project is released and the server is notified of the lock status change.

Session Management

1. Logging in.

User login to the Editor is only possible if there is no currently running session. A user is required to enter a username/password pair, which is verified at the server. If the login was successful, i.e., the username/password pair was matched at the server’s end, a new session is created and the user is authorized to use the Editor. The newly created session becomes the current session of the Editor, and the logged in user becomes the session user, as well as the current Editor user.

Login could fail because the server could not recognize the specified username, or the password provided for the specified username does not match the server’s copy of the password. The user is notified accordingly in the event of a failure, and is permitted to re-enter the username and password as necessary. Currently, a user is allowed to log in to an arbitrary number of editor instances.
2. Logging out.

Logging out of the editor is only possible if there is a current session. If there is a current project and it has not been saved since it was last modified, the user is given the option to save it. The project then is closed and the server is notified of the logout request. Upon obtaining an acknowledgement that this request has been received, the Editor's becomes inactive, and current session, user, team, project, and version all become empty. A login screen automatically appears to enable subsequent login attempts.

3. Quitting the application.

If a current session exists, the session user is automatically logged out. The application is then closed.

**Project Management**

1. Creating a new project.

Before a project can be created, the current team must be determined. If there is a current project and it has not yet been saved, the user is given an option to save it. A new project with an initially empty version is then created. The project that is currently open is closed, and the new project becomes the current project.

   The task of determining the current team could be avoided by having the user specify a team each time a new project is saved. This approach, although feasible, would require that a user select a team each time he or she executes a team-related operation, and as a result, could become burdensome.
2. Opening a version of an existing project.

Like creating a new project, this operation also requires that the current team is determined. The user is then allowed to save the current project if it exists and it has not been saved since it was last modified. Once this has been done, the user is allowed to select an existing project from the list of projects associated with the current team. Once a project is selected, the user is further allowed to select an existing version of the selected project. The possible pairs of projects and versions that can be selected are determined by querying the server. If neither a project nor version was selected, this operation is canceled. If the user has not acquired a lock on the selected project already, an attempt to grab a lock is issued. If the lock could not be acquired for some reason, e.g., a user already owns a lock on the project or a server failure, the project is opened in read-only mode and the selected version becomes ineditable. Otherwise, if the user obtained or already has a lock on the selected project, the selected version is marked for editing. Once the lock status of this project has been determined, the project and version selected become the current project and version, respectively.

3. Closing the current project.

This operation requires that a current project exists. The user is prompted to save the project if it has not been saved since it was modified, and the project is closed. The current project and version placeholders become empty.

4. Saving the current project and version.

This operation requires that the current project and version are both non-empty. If the current project has never been saved before, the user is allowed to specify a project title. If
the current version has not been saved before since it was last modified, the user is also allowed to specify a version title.\textsuperscript{2} If neither the project title nor the version title could be determined, this operation is aborted. Otherwise, the project and version is saved on the server, with the version recorded as a new version of the project, as well as the project’s most recent version. The server also records this project as locked if it has not already done so.

5. Publishing a current project.

Publishing requires that the current project, version, and lock exist. A user can publish the current project if it has been previously saved, and if he or she has a lock on the project. The user is allowed to select a version of the project. If a version was selected, it becomes the project’s only published version. Any version previously marked as published is automatically designated as unpublished. Publishing a version of the current project makes that version viewable to all members of the system via the Project Browser.

6. Browsing existing team projects.

The only requirement for this operation is that the user is associated with a team project, i.e., the user belongs on team that has at least one project. The user is presented with an interface for browsing the existing versions of all the projects he or she is authorized to modify. Authorization is determined by checking whether the user is on the access control list for the team project.

\textsuperscript{2} It is important to note that, by definition, a project that has never been saved before always has a version that has never been saved before since the project was last modified.
7. Printing the current project version.

Printing is only possible if the current project and version exist. The user is presented with a dialog for printing the onscreen project version.

8. Changing the current team.

This operation allows the user to set the team workspace used by the subsequent calls to other project management operations, e.g., open, browse, etc. If a current project exists, the user is given the option to save it. The user is then allowed to select from a list of teams that he or she is a part of. If a team was selected and it does not match the current team, the following actions happen in the specified order:

- the current project is closed;
- the current project and version both become empty;
- the selected team becomes the current team.

General editing

These operations include standard editing operations. They each require that a current project and version exist. It is important to note the following:

- Operations that modify the state of the media panel also require that a current lock exists.
- The copy operation transfers a set of media objects to the system clipboard. In addition to these media objects, contextual information, such as the size of the smallest rectangle bounding the selected objects and the location of each object relative to this rectangle's origin, are copied along with the objects. The paste operation assumes that this is the content structure of the clipboard. The content format specification is omitted here.
because it is implementation-specific. However, copied and pasted content with regards to the Editor’s use must be in the same format.

• An iconic interface is provided for some of these operations.

1. Undoing the most recent undoable action on the media panel.
   The most recent undoable operation performed is undone if it exists. This operation has the common effect of enabling the redo operation if an action was undone. The current undo depth is 100. A reasonable way of extending this implementation is to allow the user to undo an undoable operation performed at an arbitrary index in the undo history.

2. Redoing the most recent undone operation on the media panel.
   The most recent operation undone is redone if it exists. This operation has the common effect of enabling the undo operation if an action was redone. Like undo, redo can be extended to support the selection of deeper redoable operations.

3. ‘Cutting’ the current selection of media objects to the system clipboard.
   If the current selection of media objects is not empty, the copy operation is performed as specified below. The selection outlines of these objects, as well as the objects themselves, are then cleared from the screen.

4. ‘Copying’ the current selection of media objects to the system clipboard.
   If the current selection of media objects is not empty, the media objects in it are transferred to the system clipboard.
5. ‘Pasting’ the current clipboard contents to the media panel.

This operation requires that the current contents of the system clipboard consist of a set of media objects and the contextual information about these objects. These contents then are copied from the clipboard unto the media panel. The contextual information copied is translated to the media panel’s coordinates.

6. Selecting all objects in the media panel.

The set of media objects in the panel are selected if this set is non-empty.

7. Clearing the media panel.

If the set of currently selected objects in the media panel is not empty, the selection outlines of these objects, as well as the objects themselves, are cleared from the screen. Otherwise, the set of media objects in the panel is cleared if this set is non-empty.

**Media manipulation**

Some of these operations allow the user to import media of various types into an open project, and edit these media objects in a media-specific manner. It must be noted that:

- Each of these operations requires that a current project and version exist.
- The currently supported media sources are pictures, text, audio clips, and video clips.
- All media sources imported into the Editor must be at most five megabytes in length. In the case of typed text, however, the number of characters allowed in a textual object is five million, i.e., $5^{10^6}$.
- A media source with a format that the Editor cannot recognize will be represented as an icon once imported into the Editor. Subsequent requests to view the media source will
launch an application on the local machine that is capable of presenting the media source.

1. Adding a picture to the media panel.

The user is allowed to add a picture located on the file system to the media panel. Only JPEG and GIF pictures are currently supported.

2. Adding text to the media panel.

The user is allowed to create styled text, which is added onto the media panel as a text object.

There is a significant amount of extensibility considerations with regards to text support in the Editor. One important consideration concerns the modeling of the text in the editor. For example, text could be modeled in HTML, which has an adequate representation of styled text and a high degree of portability across different machines. Modeling the text in HTML however requires HTML parsing and rendering mechanisms must be available in order to view the text. Although these are mechanisms that may be available, they need to be feasibly adapted to the text-related interfaces of the Editor.

3. Adding an audio clip to the media panel.

The user is allowed to add an audio clip onto the media panel. The currently supported audio formats are AU, MP3, and WAV formats.
4. Adding a video clip to the media panel.

The user is allowed to add a video clip onto the media panel. Only videos in the QuickTime movie format are currently supported.

5. Adding the currently selected media object to the Resource Folder.

This operation requires that only one media object is selected. The user is then required to include some keywords to annotate the objects. The user is also allowed to add comments as secondary annotation. A media resource is then created using the object, and is saved on the server. If the object has already been added to the server as a resource, this save operation is prevented and the user is warned accordingly. Media resources can only be created from picture, audio, and video media objects, since those objects can be independent, to a high degree, of the projects they are included in.

6. Editing the currently selected media object.

This operation is applicable if there is only one selected media object. The user is then allowed to modify certain mutable attributes of the media object. Currently, the only objects that can be 'edited' are text objects, since other types of media are generally static. Editing a text object is similar to changing its text data.

7. Setting the key frame of the currently selected media object.

This operation is possible only when there is one selected media object. Furthermore it applies to only time-based media types, such as audio and video media. The key frame of a time-based media object refers to the start frame of the media object's source. The user is allowed to set its value, and, as a result, the project containing the media object is modified.
to always position the frame start index of the object relative to that value anytime the project is viewed.

**Media panel editing**

These operations primarily affect the state of the media objects located on the editing panel. Such state includes each media object's location relative to the panel's coordinates, its size, and its selected state. These operations require that a current project and version exist. They include the following capabilities:

- adding an object to the set of currently selected objects
- removing an object from the set of currently selected objects
- deselecting all media objects in the current selection
- dragging the current selection across the screen
- changing the size of the currently selected media object

### 4.3.1.2 Administrator

The functionality provided by the Administrator is implemented by four operation modules. The operations performed by these modules are related to sessions, users, activities, and teams, respectively. Like the Editor, all operations except login and quit require that a current session and administrator exist.

**Session Management**

1. Logging in.

Logging in to the Administrator is only possible if no current session exists. The administrator is allowed to enter a username/password pair, which is sent to the server for
verification. If username and password correspond on the server, a new session is created with the administrator as the session user. The newly created session and the user effectively become the current session and current user, respectively. Like the Editor, an administrator login attempt could fail for two reasons – either because the server could not recognize the supplied username or because password specified for the username does not match server’s copy of the password. The user is notified accordingly of either failure types, and is allowed to re-enter the username and password. The current version of the Administrator allows an arbitrary amount of login attempts. However, for reasons of consistency, an administrator can only have one Administrator application instance running. If multiple Administrator instances were permitted, the state of one instance may not correspond with the state of another, especially if the sequence of operations performed are interleaved from the server’s perspective.

2. Logging out.

An administrator can only logout of the Administrator if a session currently exists. The server is then notified of the logout request and sends an acknowledgement that it received and handled the request. The current session and administrator become empty, and the application visual state is refreshed. Finally, a login screen automatically appears to allow another administrator to log in.

3. Quitting the application.

If a current session exists, the session user is automatically logged out. The application is then closed.
User Management

This set of operations allows the current administrator to add, edit, and remove users using appropriate graphical user interfaces. User information currently consists of the user's first and last names, username, and password.

1. Adding a user to the current administrator's list.

This operation requires the administrator to specify information about the user about to be added. The only constraint on this operation is that the specified username does not already exist in the database of users. Note: Comparison between the username and the specified username is case-insensitive.

2. Editing a user in the current administrator's list.

The administrator is allowed to edit information about a user if only the user is not currently logged in on an Editor instance. All fields except the user's username are editable.

   In a future version, editing certain user information will most likely be delegated to the user, instead of the user's administrator. Such information would include the user's password information. Delegating this responsibility to the administrator has the benefit of coupling user management operations together. However, this approach is usually impractical within a widely dispersed community.

3. Removing a user from the current administrator's list.

Like the user edit operation, removing a user is only possible when the user is not logged in on the Editor. The user is disabled and hidden but not discarded. Disabling a user makes it possible to re-enable the user in a future implementation. Once a user is removed from the
administrator's list, all teams to which this user belongs are updated accordingly. The effect of this update is not critical to other simultaneously running Crossroads applications.

Activity Management

Administrators can add, edit, and remove activities using appropriate graphical user interfaces. Activity information currently consists of the activity name and a brief description of the activity.

1. Adding a new activity to the current list of activities.

Adding a new activity requires the current administrator to specify the necessary activity information. The only constraint in this operation is that the name of the new activity cannot match the names of any existing activities. Note: String comparisons to enforce this constraint are case-insensitive.

2. Editing an activity.

The only editable information about an activity is its description. The administrator can edit this information at any time, regardless of the state of any currently running Editor instances.

3. Removing an existing activity.

Removing an activity is only possible if each team within the activity is not currently active in any running Editor instance. Upon removal, all team information about this activity is disabled and hidden but not discarded. The effect of this removal is not critical. An activity is currently active if the current team of any simultaneously running Editor instance is.
associated with the activity. A future implementation will allow the administrator to enable previously disabled information about an activity.

Team Management
Within an activity, administrators can add and remove teams. Administrators can assign users on their list into a particular team, and also remove specific team members. The number of members any team can have is arbitrary. Teams within an activity may also contain duplicate users. Appropriate graphical user interfaces are provided to allow the administrator to add and remove teams within a particular activity, and to assign a user to a particular team, and remove a user from a specified team. Team information currently consists of only a name, which is uneditable.

1. Adding a new team within an activity.
Adding a team requires the administrator to specify a name to denote the team. The team is only registered with the activity if a name conflict does not arise, i.e., no team of the same name exists within the activity space. If there is a name conflict, the administrator is accordingly notified. Note: A case-insensitive name-matching algorithm is used to check for conflicts.

2. Removing a team within an activity.
A team can only be removed if the team is not a current team within any simultaneously running Editor instance. Once a team has been successfully removed, all information about its projects is disabled and hidden but not discarded. Disabling a team makes it possible for the administrator to re-enable the team in a future implementation.
3. Assigning a user to a team.

The user being assigned to the team must be selected from the administrator's list. Only users not currently in the team can be assigned to the team. This operation can be performed at any time, since it does not affect the members currently in the team.

A likely extension to this feature would be to support the notion of a master administrator. A master administrator would be a special type of administrator who has been granted the permission to add users from one administrator's list to another administrator's list.

4. Removing a user from a particular team.

This operation has the distinction of being the lowest-level mutability operation with constraints. It cannot be performed if the current user is active in a currently running Editor instance. The effect of this operation is not critical to other client applications.

4.3.1.3 Resource Folder

The Resource Folder's implementation includes support for browsing and searching through media resources of various types and commenting on a particular media resource.

1. Browsing through available media resources.

In order to support browsing, the Project Folder maintains a cached copy of the list of media resources currently available in the system. If the cache is empty, this list is retrieved using a network connection. If this connection is available to the Resource Folder, it is used.
Otherwise, a new connection is obtained. Once the list has been retrieved, it is presented in the browsing interface. The resources included in the list are categorized by type within this interface for easier identification, and the user can select a resource from this list for viewing.

2. Searching through available media resources.

In order to search for media resources, the user must provide a query string containing keywords related to the media resources they expect to find. Optionally, the user can also specify the set of available media types to search through. The default set includes all media object types. The search is performed on the server, and a list of matched media resources, if any, is returned and presented in the search interface. The user will then be able to select a media resource in this list for viewing.

3. Displaying a media resource.

This operation occurs whenever the user selects a media resource for viewing. The selected resource becomes the current resource, and is shown in the media panel.

4. Refreshing the current view of the resource folder.

Refreshing has the effect of clearing the cache of media resources maintained by the application and retrieving an updated copy from the server. This copy is then cached and presented in the browsing interface. The search interface need not be modified.

An alternative implementation would be to make the necessary resource changes (if any) to the current browsing view, as opposed to clearing this view and updating it with the resource collection. This implementation is more efficient since relies on the fact that any

---

3 A network connection is will be available if the Resource Folder was opened within a currently running instance of the Editor or the Administrator.
object that becomes a media resource will not be removed. Its keywords and comments attribute may change, but the name, creator, and media source of the resource will remain constant. Thus, the only kind of resource changes that is possible is the addition of new resources. This implementation was sacrificed for scheduling reasons, and a future version of the Resource Folder will incorporate it.

5. Adding comments to the selected media resource.

Comments may include keywords associated with the resource, a short amount of descriptive text, etc. The comments provided are added to the collection of comments attached to this resource on the server.

6. Copying the media contents of the current media resource to the system clipboard.

This operation resembles the copy operation provided by the Editor. The user is allowed to transfer the media resource contents to the clipboard for use in other applications, e.g., the Editor.

4.3.1.4 Project Browser

The implementation of the Project Browser bears a close structural resemblance to that of the Resource Folder. The operations in this implementation consist of the following:

1. Browsing through available publishable projects.

Like the Resource Folder, the Project Browser maintains a local cache of the view of the collection of publishable projects. The browsing interface relies on the contents of this cache for presenting the projects, each of which is identified by its project title. If the cache
is empty, the current set of publishable projects is retrieved from the server, and the cache is updated. Any project selected from the browsing interface becomes the current project, and the contents of the media panel are replaced with the media contents of the published version of the project.

2. Searching through available publishable projects.

A user can search for published projects by specifying a query. A title search is then conducted on the server, and a list of all projects whose titles contain any of the words in the search query is returned and presented to the user through the search interface. The user can select a project from this list for viewing on the media panel.

Several extensions can be added to the provided implementation. One extension would be to allow the user to specify additional searching parameters, such as user, team, or activity information. Using these parameters, a user might be able to find projects related to a particular user, team, or activity, and not just projects whose titles match the words in the search query. Another extension would be to require that published projects contain a brief textual description of the project, or the idea or story presented by the project. Using this description, deeper and higher quality searches than simple title search can be performed. Although neither extension was implemented in the current version of Crossroads because scheduling constraints, they will be considered in future versions.

3. Displaying the contents of a project.

The contents of the published version of a project are displayed in the media panel whenever the user selects this project. The selected project also becomes the current project.
4. Refreshing the current view of the browser.

The cache containing the set of publishable projects is cleared and the current set of published projects is retrieved from the server. These projects are placed in the cache, and the application’s browsing interface is updated with the project listing.

Instead of causing a complete listing update, this operation can be modified to update the project listing with only those publishable projects that are new or have changed versions since the listing was previously retrieved from the server.

4.3.1.5 Failures
In the event of a client application failure, an appropriate message is displayed to the user, and the operation currently being performed is aborted. Such failures are usually attributed to the inability to locate or access a resource, e.g., the file system or a server. User errors, e.g., leaving a project title field blank during a project save operation in the Editor, are also shown to the user, and in most cases, the user is allowed to retry the corresponding operation.

4.3.1.6 Performance
In general, all client applications exhibit the same performance. They use the intranet infrastructure as leverage to obtain a relatively high degree of speed when accessing the Crossroad servers. With the exception of an application’s current state, most data used by the application is stored on the server, thereby freeing most secondary storage resources used by the local machine on which the application is running. The secondary storage that is maintained on this machine consists mainly of media source data that is used by the client applications running on that machine.
By adopting the idea of a current set of data during their execution, these applications exhibit an efficient management of primary memory. When the current data changes, the memory allocated for previously referenced data objects is released. Since the data released is not cached, some speed loss might be incurred. However, in an effort to maintain low storage cost on each client using the system, this loss is not a significant problem.

4.3.1.7 Potential Modifications
One likely and relatively easy modification to the client applications that provide viewing of project and media content is to enable print support whenever such content is displayed. This implies that the Project Browser and Resource Folder implementations would be extended to allow the user to print the current project and resource, respectively.

4.3.2 Server Applications
4.3.2.1 Standard Server
The implementation of the Standard Server significantly relies on a connection to the central database. The server uses this connection to retrieve information from the database, as well as update this information when requested. The server also performs necessary processing tasks, mainly input and output formatting, before and after it communicates with the database.
4.3.2.2 Administration Server
The Administration Server implementation is similar in structure to that of the Standard Server. The Administration Server's adds administrative functionality to the base functionality provided by the Standard Server, e.g., adding users and teams.

4.3.2.3 Failures
Server failures are usually attributed to database access errors. In event of such situations, the client of the server operation during which the error was generated is notified accordingly and the operation is aborted. Errors generated as a result of an invalid user input will trigger a specific exception, which is forwarded to the client of the operation. The server operation will then be terminated.

4.3.2.4 Performance
Like the client tools, the network communication with a server should be relatively fast if the clients and server are located within the same intranet. Besides this reasoning, the server implementation demonstrates at the minimum a moderate level of performance.

The general performance of any Crossroads server is highly dependent on its access to the database, and the database’s performance. Communicating with the database should be relatively fast if the database is located within the intranet as recommended. In addition, the database used in the implementation may be optimized for acceptable performance in some cases.

4.3.3 Resources Used
The design of all Crossroads applications is implemented using the Java Development Kit 1.1.8. Java was chosen as the implementation language primarily because it is highly portable.
across various computer platforms. It also offered several benefits to the development of
the system, such as automatic memory management, strong type checking, etc.

The Java Foundation Classes (JFC) 1.1 with Swing 1.1.1, commonly known as Swing, was chosen over the standard Java Abstract Window Toolkit (AWT) classes to implement the graphical user interface for the client applications. Using the JFC over the AWT made it possible to provide more advanced graphical user interfaces, which offered significantly more functionality than AWT components. In addition, since Swing components are written entirely in Java, they work similarly on every platform, thereby increasing system portability. An alternative implementation may rely on the AWT classes in order to achieve better performance since the AWT uses the resources of its native windowing system, as opposed to Swing components. However, a substantial amount of additional development work will need to be done.

Since the standard Swing components, or AWT components for that matter, do not provide support for time-based media such as audio and video media, the QuickTime for Java 4.1 (QTJ) library is used to adapt these media formats into a representation that is usable by Swing components. Once this adapting is done, standard Swing components can be used to present and manipulate a video clip. Java Media Framework (JMF), a Java extension framework provided by Sun, was considered as an alternative for implementing time-based media presentation. However, JMF posed several constraints, such as a heavy dependence on native windowing resources, thus making it difficult to integrate with Swing components.

Since both client and server applications were going to be written in Java, a specific network communication paradigm specific to Java was considered. Java provided two standard network mechanisms: sockets, which provided basic networking support, and
remote method invocation (RMI), which allowed programs in one Java Virtual Machine to call programs in another Java Virtual Machine through the use of interface methods. Both mechanisms proved to be suitable for the design. However, RMI was chosen over sockets because it was a more convenient abstraction to use. If offered a model of the network communication at a level higher than that provided by sockets, which was highly desirable for the system. Alternative implementations of the design may have used more advanced networking frameworks, such as CORBA. However, the server implementation may have to change in other to support these frameworks.

A MySQL database was used as the central database. MySQL was chosen because it provided considerable savings in cost and speed than other standard databases. It also offered a high degree of reliability, and had received tremendous support from its user community. The MM MySQL JDBC driver was used as a bridge between the server applications and the database. The driver was one of a few officially recommended by the MySQL designers. Although MySQL offered several benefits in performance and speed, it lacked a rigid transactional mechanism, which prompted us to sacrifice some system usability in the implementation in order to achieve data consistency. Approaches that can work around this discrepancy will be considered in a future system implementation.
5 PRELIMINARY EVALUATION

In the present chapter, we will present the results of our preliminary evaluation of Crossroads—which primarily consisted of a few detailed case studies of the system's use by a small number of potential users (students, teachers, museum educators, and after-school mentors) as well as our colleagues and outside critics while the system was being developed. These studies not only have shaped the current design of the system but also have given us a better sense of how to evaluate the system in the next rounds.

In our case studies, we decided to get an overall sense of a user’s opinion of the system functionality. This opinion would include thoughts mainly about the use of the system, and about how adequate the system is in supporting collaborative authoring. Although our ideal target was to try out the system at a few schools and after-school centers since they were true educational contexts, it proved impractical to do along with the design and implementation in a one-year Masters program (These types of evaluations are under development in the planning stages now). As a result, we decided to have fellow researchers at the MIT Media Laboratory evaluate the system intensively for about a week or two. The researchers would be chosen carefully so that they could provide us with valuable feedback about the system’s applications.

The study consisted of five participants, mostly graduate students from the laboratory. Each student had participated in an activity that they had not yet documented, and many of them decided to document their projects in the activity with the Crossroads Editor. They spent about a week working on their projects. Two of the participants were placed in a team, while the others worked individually.
Owing to the system’s infancy, the projects created by the participants were relatively small. However, a significant amount of feedback was received from the study. This feedback was mostly related to the functionality supported by the tool. It included suggestions for better pagination support, improved clipboard operations, advanced text support, image resizing, and local storage support. In terms of a collaboration support, the tool was praised for enabling both individual and collaborative authoring, especially its automatic versioning feature. The remaining feedback was related to the type of settings the system would be most useful for, which include collaborative authoring settings and multimedia presentations. A participant also pointed out that the system may be modified slightly to become a useful document layout program with support for drawing lines and shapes.

Educators from museums and other institutions also provided feedback regarding Crossroads at research open houses at the Media Laboratory where the system was presented. This feedback was generally positive, including praises for the system’s goal of enabling collaboration in educational contexts as well as the editing capabilities it offered.

An important lesson learned from the case studies was that an evaluation of Crossroads requires careful thought and structure. This is necessary because the tools provided by Crossroads are developed for an existing community of users, which may be small or large in number, and it is our belief that effective and constant use of these tools will facilitate the growth of this community. This growth can be modeled in various and sometimes unexpected forms, and a well-structured and comprehensive evaluation of Crossroads’s use will help shed light on some of these forms. In addition, since Crossroads is designed to be used regularly over periods of time, patience would be required from the case study participants.
We intend on conducting the studies described above since the system implementation is currently in progress. Specifically, we are planning to conduct short-term studies of the system's use in a variety of real settings once system functionality has been tested thoroughly. Some of these settings include groups of students in a single classroom, in a cluster of multiple classrooms, and in an after school center. We are also planning to study Crossroads' use by the group of teachers and curriculum activity developers in these settings, and determine the implications it has for them. From these studies, we hope to learn to what extent the project library can act as an inspiration provider, enhance our understanding of how technologies can be applied to learning environments, and also determine how effectively the functionality provided by Crossroads supports collaboration in these environments. These results will serve as an invaluable guide in considering the future directions of Crossroads.
6 FUTURE WORK

As mentioned earlier, Crossroads as a system is a work in progress. Extensions to the current version of the system include a more portable representation of project content, more collaboration features such as chatting and WYSIWIS (what-you-see-is-what-I-see) editing, support for additional administrative features such as administrator-to-client feedback and remote synchronization of project content, and more efficient deployment of multiple servers and databases. Additionally, the system, mainly in terms of its component tools, is planned to undergo rigorous testing and evaluation. In the remainder of this section, we will discuss each of these issues in detail and close with some concluding remarks.

Representing project content

A project in Crossroads consists of a collection of media objects, and a context that references these objects in a specific way. Currently, this context includes information such as the onscreen location and dimensions of each media object, as well as the visual ordering of the media objects for handling overlaps. The context maintains additional information for an audio or a video media object, such as its key frame number. While this contextual information captures useful information about the media objects in a project, it is fairly simple in representation and lacks some richness. Ideally, contextual information may not comprise only of information about the media objects; it may include the background color of the project space, for example, and other kinds of visual elements such as shapes and lines that are not related to standard media. Thus, it might be beneficial if Crossroads was extended to support more flexible media representations.

Another source of future extension related to project content concerns how the content is modeled in a Crossroads implementation. Project content in Crossroads is modeled completely in
Java, and as a result a Java Virtual Machine is needed to view the project. Since project content currently consists of objects that reference multimedia elements (pictures, text, audio, video) that can be displayed in HTML, an alternative way of modeling project content is by using HTML instead of Java. Specifically, it would be beneficial to have a mechanism that converts a Crossroads project to simple HTML, without a huge loss in the project's original formatting. Such a facility would enable projects to be viewed conveniently by many, as long as a Web browser is available. Although Web browsers might need to have the necessary resources installed, e.g., plug-ins for viewing video data, for example, presenting a Crossroads project over the Web would attract a wider audience than doing the same through a Java program. Additionally, more capabilities are enabled by taking advantage of this conversion utility. For example, in order to print an onscreen project, it would be more beneficial and less error-prone to the software developer to convert the project to HTML and automatically or manually have the HTML output sent to a printer, than having to write code to handle the printing necessities. Although the conversion could occur both ways, it would be conceivably more challenging to parse the contents of a Crossroads project from HTML, since HTML is not usually as expressive as Java. This task becomes even more challenging if the project representation is extended to support new features. Thus, this conversion is discouraged for the sake of retaining the original formatting of the project content.

As an alternative approach to making project content portable is to retain its Java object representation and provide an applet for viewing the content in a Web browser. Such an approach is highly beneficial, since the project content is kept intact when viewed. The primary disadvantage of this approach, however, is that applets take a noticeable length of time to load, which usually scales proportionally with the size and number of Java classes they use. The current implementation uses a considerable number of objects to display media content, particularly audio and video content, and this number is likely to affect the perceived response time for viewing the project. As a
result of its benefits and drawbacks, Java applet support will be considered thoroughly in a future system design for presenting media content.

**Storing data locally**

All data used by the Crossroads client applications is maintained remotely. The only part of this data that is also stored on a user’s local machine is the collection of media sources that have been used by previous client application instances on that machine. A connection to a Crossroads server is needed to access all other information, which might be impractical or unavailable in some settings. Thus, it would be highly advantageous to maintain some Crossroads data locally during or after running a client application. For instance, it would be useful to browse through lists of projects and media resources in the absence of a network connection. Once a network connection is available, this list and other locally stored data can be synchronized with the server to ensure data consistency. Local data storage also has the clear advantage of increasing the usability of the system in certain cases. In addition, having data stored locally would generally result in a shorter response time to access that data than if the data was stored on a server. Owing to these benefits and other significant ones, local storage of selected Crossroads data will be incorporated into a future system design.

**Supporting additional collaborative features**

Currently, collaboration within a team consists of viewing and updating the work done by a particular team member. This collaboration design does not, however, enable team members simultaneously using Crossroads communicate with one another, perhaps about a particular project. This missing functionality can be added by incorporating chatting facilities to the system design. Using chat, team members will be able to communicate over long distances, regarding, for example,
how to proceed on a particular project or anything they wish in general. Fortunately, the current
design allows this addition to be made easily. The chatting module can be designed independently
of the existing system components and can be embedded in a helper client tool that can be run
independently, or started from top-level tools such as the Editor or the Administrator. Starting the
chatting program from either top-level tool requires a little modification to the implementation of
the tool. Owing to the large volume of Java-based chatting programs, chatting support may even be
obtained from an existing, sophisticated application instead of being designed from scratch. A
useful source of chatting application research is work of Judith Donath, an Associate Professor at
the MIT Media Laboratory and head of the laboratory’s Sociable Media Group. Of particular
interest to the Crossroads effort is Chat Circles [21], a Java-based graphical user interface application
developed by members in her group. Chat Circles not only enables chat within members in a
community, but also uses shapes and forms to convey the identity of its users and their chatting
activity intuitively.

Another extension to Crossroads closely tied to chat support is WYSIWIS support. Using
WYSIWIS, team members can view the work being done by the lone team member who currently
has write access to the work. When applied to Crossroads, this feature serves as an enhancement to
the experience of the team member who does not currently have write access to a particular project.
Instead of having a user reopen a locked project to view its current contents, Crossroads can use
WYSIWIS to display to the user the contents of the projects as it is edited by the user who owns the
write lock. Thus, a user who cannot edit a team project can view the project’s progress – in real
time. In order for this mechanism to work, however, a server needs keep track of the users logged
in to Crossroads at a particular point in time, and the location of the machines that they are using.
This information is already maintained by the current design of Crossroads; the only task that is
necessary is to design the transfer of project content as it is being edited and the presentation of the
content on the running instances of the Editor that are authorized and interested in viewing the content. Once implemented, WYSIWIS editing could be integrated with chat support to provide a true and intuitive feel of collaborative authoring.

Providing more administrative-level features

Monitoring user activity is a common task carried out by administrators. In schools, for example, teachers are usually responsible for monitoring the progress of their students in the various activities or projects that the students are involved in. In certain cases, it might be worthwhile if such capabilities were translated to Crossroads. Specifically, it might prove beneficial to add a set of features to the Administrator that will allow administrators view a log describing the Crossroads activity of their users. These features might enable teacher to determine exactly which students are collaborating together and how they are collaborating. They could also help to illustrate individual habits and interests such as media preferences that might be hard to observe otherwise. An issue regarding the support of this feature is exactly what kinds of activity information to log for future viewing, which could vary depending on the administrator. They could range from the simple (Editor login and logout times) to the elaborate (editing operations performed). As the current implementation of Crossroads logs all user and administrator sessions in the central database, designing capabilities that, at the least, allow an administrator view the appropriate segment of this log seems like a reasonable extension to support.

In addition to the possibility of providing user-monitoring support, it might also be worthwhile to provide capabilities to Crossroads that will facilitate dialogue between an administrator and a user. Such capabilities may have a noteworthy impact in schools where Crossroads is installed, since they serve as convenient channels for teachers to provide valuable feedback to their students as they work with others. Students, and the teams they work in, can use
this feedback as a resourceful guide in their efforts. Since it might also be the case that the teacher needs to communicate with a particular student, using this feature would help facilitate such communication in a manner that is sometimes more explicit than oral communication. In spite of these benefits, however, this feature might be destructive to collaborative learning if not used effectively, particularly within constructionist-based activities. For example, a teacher might constantly use the tool to direct a student on his or her work, thus taking up much of the capacity that the student could have used to explore and experiment on his or her own. As a result, this issue and others related to it need to be carefully considered in any design supporting communication between administrators and users.

Additional functionality that could be incorporated into a future design of the Administrator includes the synchronization of Crossroads content stored locally within an intranet with a designated server at a remote location. Once stored at a remote server, such content be later retrieved in the event of a data transmission failure or a database corruption within the intranet. Some information loss might occur in this retrieval process, depending on the frequency of the synchronized updates. As an alternative, this synchronization feature can be designed completely in the background, thus relieving the administrator of several responsibilities. While this feature can provide tremendous benefits, it can be relatively complex to design since factors such as Internet connectivity and network bandwidth need to be accounted for.

A final modification to the Administrator design would include support for creating project templates. Using templates, teachers, for example, can ensure that projects created by their students in particular activity conform to a particular presentation format. This standardization can result in various benefits for the administrator, particularly in reviewing the project. It could also serve as a presentation guide for the users in publishing their work. However, like the administrator-to-user communication functionality discussed earlier in this chapter, this feature could limit in the learning
process of the user, given that templates are being designed for constructionist activities. Specifically, the templates can act as a restriction on an individual's expression of an idea, and thus, may suppress the development of an idea rather than promote it. As a result, the support of templates in future Administrator designs needs to be considered with caution.

Using multiple servers and databases

Within an intranet, it is possible to deploy multiple Standard and Administration Servers, since both kinds of servers are sharing the same database, which is also situated within the intranet. However, when using additional servers outside the intranet to increase availability, it becomes necessary to configure these servers with the database in the intranet. This task may be difficult or impossible, since each server is designed to currently access one particular database and some of these servers may not have been configured for the intranet.

In such a situation, it might become necessary to combine database replication schemes with server implementations that are parameterized over the set of databases that they can connect to. Using this approach, it becomes possible for client tools to access the appropriate data from using its connection to an authorized server, regardless of whether or not the server is located within the intranet. Although this approach is more robust than the current approach, it is considerably more complex to design and can be costly. As a result, in order to increase availability and flexibility, future system implementations need to consider this approach, or adopt variants of it that are less costly and simpler.

Testing and evaluating the system

Crossroads is slated to undergo comprehensive testing procedures such as the formal documentation of all testing modules, the creation of a varied suite of testing samples, and the
recording of all testing results. Also, as it is being developed, the system will be evaluated periodically in various contexts to obtain feedback about its usefulness both as a collaborative tool and as an authoring tool.
7 CONCLUSION

The increasing use of the Internet has led to efforts in discovering ways to facilitate collaboration in various institutions in society. International conferences such as the CSCW and CSCL conferences are testaments of this fact. While these conferences have promoted the development of software tools such as QuickPlace™ that are designed for the collaborative work in general, they have also fostered discussion on the impact that these tools can have on collaborative learning. Such discussion has recently led the way to the introduction of creative software tools targeted towards the educational community at large, which help to bridge the gap between various modes of collaborative learning and technological possibilities.

This thesis has described the development of Crossroads, an extension to the small number of collaborative software tools designed for educational community. Unlike many of its previous counterparts such JavaCAP and Belvedere, Crossroads can be used to support a diverse array of collaborative activities, particularly those based on the ideals of constructionism. In other words, Crossroads could be effectively adopted into a community that is structured around the perspective of the individual as someone who can be given enough capacity to explore his or her own understanding of a particular concept and present this understanding to others in a form that is tangible, self-created, and meaningful. Through this presentation, members of the community can share with each other the efforts and results of their work. They are also able to reflect on the relevance of their work from the feedback received from others, as well as their personal interpretation of their work throughout the construction process.

Crossroads promotes this learning paradigm through the use of the several tools it offers. These include a flexible editing program that allows users to document the planning, process, and results of their work in a given activity, as well as an administrative tool for use by administrators.
who monitor the progress of the users' work. Other tools for browsing other users' projects and sharing media with other users help to facilitate collaboration among the users of the system.

While the Crossroads effort is promising, preliminary case studies have illustrated that more work needs to be done. Such work includes designing a more portable way of representing project content, integrating chat support into the system to provide additional avenues for team collaboration, and extending the set of administrative capabilities that are currently available. With the successful outcome of this work, it is probable that Crossroads will eventually become a suitable software platform for facilitating collaborative learning in educational environments.
ACKNOWLEDGEMENTS

Foremost, I would like to thank Bakhtiar Mikhak for giving me the opportunity to be involved in Crossroads and for directing me this effort. I am indebted to my family and my friends, who supported me in various ways throughout my work on this project. Many thanks to the members and friends of Epistemology and Learning Group at the MIT Media Laboratory, particularly Rahul Bhargava, Robbin Chapman, Teresa Hung, Daniel Kornhauser, Fred Martin, Mitchel Resnick, Casey Smith, Carolyn Stoeber, Sam Thibault, Diane Willow, and Alice Yang, as well as several Cube dwellers.
BIBLIOGRAPHY


5. CSCW '98. http://www.acm.org/sigchi/cscw98/


10. LEGO MINDSTORMS. http://mindstorms.lego.com/


APPENDIX

NOTE: The abstraction specifications below are not final, and are subject to change. This is because Crossroads is currently in development. However, the provided specifications are the most recent as of May 23, 2001, the time of submission of this thesis.

For the most recent specifications, please send email to coolfash@media.mit.edu.

A APPLICATION ABSTRACTIONS

A.1 Editor abstractions

A.1.1 CrossroadsEditor

class CrossroadsEditor {
    Overview: CrossroadsEditor is the target abstraction for the Editor application. It maintains a set of EditorManagers, which are responsible for most of the Editor’s functionality. This set includes a FileManager, an EditManager, a MediaManager and a ToolsManager. CrossroadsEditor also provides functionality for session login and logout, as well as interfaces for EditorManagers to update the Editor's state.

    void login()
    void logout()
    void quit()
    void exit(int)
    FileManager getFileManager()
    EditManager getEditManager()
    MediaManager getMediaManager()
    ToolsManager getToolsManager()
    Session getCurrentSession()
    ServerConnection getConnection()
    MediaPanel getDisplayPanel()
    Team getCurrentTeam()
    Project getCurrentProject()
    ProjectVersion getCurrentVersion()
    ProjectVersion getOnscreenVersion()
    void setStatus(String)
    void setCurrentTeam(Team)
    void setCurrentVersion(ProjectVersion, int)
    void setCurrentVersion(ProjectVersion, int, boolean)
    void setCurrentProjectAndVersion(Project, ProjectVersion, int)
    void setCurrentProjectAndVersion(Project, ProjectVersion, int, boolean)
    boolean isCurrentVersionModified()
    boolean isCurrentProjectEditable()
}
A.1.2 FileManager

interface FileManager extends EditorManager{
    Overview: The FileManager interface (for the CrossroadsEditor) specifies a collection of
    operations related towards the management of projects, such as the creation, opening, and
    saving of projects.

    void newProject()
    void openProject()
    void closeProject()
    void saveProject()
    void browse()
    void publishProject()
    void printProject()
    void changeTeam()
    void login()
    void logout()
    void quit()
}

A.1.3 EditManager

interface EditManager extends EditorManager {
    Overview: The EditManager interface specifies a set of operations concentrated towards
    general editing of a project's contents.

    void undo()
    void redo()
    void cut()
    void copy()
    void paste()
    void clear()
    void selectAll()
}

A.1.4 MediaManager

interface MediaManager extends EditorManager {
    Overview: The MediaManager interface specifies arbitrary media manipulation operations.
    These include the addition of new media and updating of media attributes. The MediaManager
    also enables media sharing.

    void addPicture()
    void addText()
    void addAudio()
    void addVideo()
A.1.5 ToolsManager

interface ToolsManager extends EditorManager {
    Overview: The ToolsManager interface provides access to helper Crossroads tools such as the Resource Folder and the Project Browser.
    void openResourceFolder()
    void openProjectBrowser()
}

A.2 Administrator abstractions

A.2.1 CrossroadsAdministrator

class CrossroadsAdministrator {
    Overview: The CrossroadsAdministrator abstraction is the target abstraction for the Administrator application. It supports several operations, which were initially designed to update information about an administrator’s users and the teams that these users are involved.
    void login()
    void logout()
    void quit()
    void exit(int)
    FileManager getFileManager()
    ToolsManager getToolsManager()
    Session getCurrentSession()
    User getCurrentUsers()[]
    ServerConnection getConnection()
    void addUser()
    void editUser()
    void removeUser()
}

A.2.2 FileManager

interface FileManager extends AdministratorManager {
    Overview: The FileManager interface (for the CrossroadsAdministrator) specifies standard session operations. In addition, it also provides operations for setting up the activity-team hierarchy in Crossroads.
    void setupActivities()
    void login()
void logout()
void quit()

A.2.3 ToolsManager
interface ToolsManager extends AdministratorManager {
    Overview: The ToolsManager interface used by the CrossroadsAdministrator is identical to that used for the CrossroadsEditor. See A.1.2 in the appendix for more details.

    void openResourceFolder()
    void openProjectBrowser()
}

A.3 Resource Folder abstractions

A.3.1 CrossroadsResourceFolder
class CrossroadsResourceFolder {
    Overview: The CrossroadsResourceFolder serves as the key abstraction for the Resource Folder application. It allows users to perform specific operations on existing media resources. These include annotating a particular media resource and searching through the list of available media resources.

    void attach()
    void remove()
    void editKeywords()
    void editComments()
    void search(String, int)
    void display(MediaResource)
    void exit(int)
}

A.4 Project Browser abstractions

A.4.1 CrossroadsProjectBrowser
class CrossroadsProjectBrowser {
    Overview: The CrossroadsProjectBrowser is the target abstraction for the Project Browser tool. It provides user functionality for browsing and searching through all team projects that have been published.

    void exit(int)
    void search(String, int)
    void display(Project)
}
A.5 Standard Server abstractions

A.5.1 CrossroadsStandardServer

class StandardServer {
    Overview: The StandardServer is the application abstraction for the Crossroads Standard Server. It provides operations for starting an instance of this server and obtaining a connection to the server once started.

    void start()
    ServerConnection getConnection()
}

A.6 Administration Server abstractions

A.6.1 CrossroadsAdministrationServer

class AdministrationServer {
    Overview: The AdministrationServer abstraction is the administrative counterpart to the CrossroadsStandardServer abstraction. While the CrossroadsStandardServer regulates user network connections, a CrossroadsAdministrationServer handles administrative network connections.

    void start()
    ServerConnection getConnection()
}

B UTILITY ABSTRACTIONS

B.1 Core data abstractions

B.1.1 User

class User {
    Overview: The User abstraction models a standard Crossroads user – a principal who can run only the Editor, the Resource Folder, and the Project Browser tools. This abstraction maintains the name, username, and password associated with this principal.

    User(int,String,String,String)
    User(int,String,String,String,String)
    int getUserID()
    void setPassword(String)
    String getFirstNames()
    String getLastNames()
    String getUsername()
    String getPassword()
    String getOfficialFullName()
}
String getAlphabeticalFullName()
}

B.1.2 Administrator
class Administrator {
    Overview: The Administrator abstraction is similar to the User abstraction, except that it is used to represent administrators instead of users.

    Administrator(int,String,String,String)
    Administrator(int,String,String,String,String)
    int getAdministratorID()
    void setPassword(String)
    String getFirstNames()
    String getLastNames()
    String getUsername()
    String getPassword()
    String getOfficialFullName()
    String getAlphabeticalFullName()
}

B.1.3 Session
class Session {
    Overview: A Session is a read-only abstraction that designates a user or an administrator as logged in to the Editor or to the Administrator, respectively.

    Session(int,User)
    Session(int,Administrator)
    int getSessionID()
    User getUser()
}

B.1.4 Activity
class Activity {
    Overview: An Activity object is used to represent an activity that Crossroads facilitates. This object maintains information about the activity such as its name, a short description of it, and the name of the administrator who setup on the activity using the Administrator.

    Activity(int,String,String)
    void setDescription(String)
    void setAdministrator(Administrator)
    int getActivityID()
    String getName()
    String getDescription()
    Administrator getAdministrator()
B.1.5 Team

class Team {
    // Overview: A Team object stores important information about a team. This information includes the name of the team, the members in the team, and the activity that this team is a part of.

    Team(int,String)
    void setMembers(Vector) 
    void setActivity(Activity) 
    int getTeamID()
    String getName() 
    Vector getMembers() 
    Activity getActivity() 
}

B.1.6 Project

class Project {
    // Overview: The Project abstraction is used to model a Crossroads project. This abstraction maintains title of the project and the team that the project is a part of.

    Project(int,String)
    void setTitle(String)
    void setTeam(Team)
    int getProjectID()
    String getTitle()
    Team getTeam() 
}

B.1.7 ProjectVersion

class ProjectVersion {
    // Overview: The ProjectVersion abstraction is designed to capture the essence of the version of a project. It stores necessary information about the project version, particularly its title, the user who edited it, the date these edits were committed, a number to distinguish this version from other versions of the same project, and media content used in the version.

    ProjectVersion(String)
    ProjectVersion(String,User,Date,int)
    void setTitle(String)
    void setAuthor(User)
    void setDate(Date)
    void setNumber(int)
    void setProject(Project) 
}
void setMediaContent(MediaContent)
String getTitle()
User getAuthor()
Date getDate()
int getNumber()
Project getProject()
MediaContent getMediaContent()

B.1.8 MediaContent
class MediaContent {
    Overview: The MediaContent abstraction is used to capture the structure of the content
displayed in a media panel. This content is composed of two types: the collection of media
sources referenced in the content, and a context that arranges these media sources in a
meaningful way.

    MediaContent()
    MediaContent(Vector,Vector)
    Vector getMediaSources()
    MediaContext getContext()
    void update(MediaSource, MediaSource)
}

B.1.9 MediaSource
class MediaSource {
    Overview: A MediaSource object is used to represent a source of media, whose data stored in
a file in a certain format. The object maintains useful information about the source, particular
such the type of the source, the size of its data contents, and the format of these contents.

    MediaSource(int,int,int,String)
    int getMediaSourceID()
    int getSize()
    int getType()
    String getFormat()
    int getTypeAsInt(String)
    String getTypeAsString(int)
}

B.1.10 MediaContext
class MediaContext {
    Overview: The MediaContext abstraction represents a context for media sources, particularly
those in a MediaContent object. This context currently consists of a list of the properties of
media objects referencing media sources in the MediaContent object.
B.1.11 MediaResource

class MediaResource {
    Overview: A MediaResource object is used to model media resources. Media resources are
    media sources that have been designated as public by a Crossroads user. A MediaResource
    object maintains information such as a name of the media resource and the annotation
    associated with it, as well as the creator of the resource. The resource creator is the user who
    added it to the Resource Folder.

    MediaResource(MediaSource, String, String, User, Administrator, Date)
    void setName(String)
    void setKeywords(String)
    void setComments(String)
    MediaSource getMediaSource()
    String getName()
    String getKeywords()
    String getComments()
    User getCreator()
    Administrator getAdministrator()
    Date getCreationDate()
}

B.2 Server data abstractions

B.2.1 DatabaseAccess

class DatabaseAccess {
    Overview: The DatabaseAccess abstraction specifies the several operations that a Standard
    Server can perform on a given Crossroads-configured database. These operations range from
    retrieving version information about a particular project to saving the contents of a new project.

    DatabaseAccess(String)
    User getUser(String)
    Session loginUser(String, String)
    void logoutUser(Session)
    Activity getActivities(User)[]
    Administrator getAdministrator(Activity)
    Team getAnyTeam(User, Activity)
    Team getTeams(User, Activity)[]
    Team getTeams(User, Activity, boolean)[]
    Project getAnyProject(Team)
    Project getProjects(Team)[]
Project getProjects(Team, boolean)[
VersionInfo getVersionInfo(Project)
MediaContent getMediaContent(Project, ProjectVersion)
byte getData(MediaSource)
void acquireLock(Project, Session)
void releaseLock(Project)
int saveProject(Project, ProjectVersion, MediaContent, MediaDataManager, Team, Session)
int saveMediaSource(MediaSource, MediaDataManager)
void publishVersion(Project, ProjectVersion, boolean)
Project getProject(String, Team)
ProjectVersion getVersion(String, String, Team)
Team getAnyTeam(User)
Team getTeams(User)
Team getTeams(User, boolean)
Project getAnyProject(User)
Project getProjects(User)
Project getProjects(User, boolean)
Project getAnyProject(User, Activity)
Project getProjects(User, Activity)
Project getProjects(User, Activity, boolean)
Activity getActivities(Administrator)
Team getTeams(Activity)
User getTeamMembers(Team)
Activity getActivity(String, Administrator)
Team getTeam(String, Activity)
int getVersionCount(Project)
int getLastInsertionID(String, String, String)
String encode(String)
String decode(String)
String getString(Date)
Date getDate(String)
String getPassword(User)
Vector getAdministrators()
Vector findProjects(String, int)
Vector getPublishableProjects(Activity)
Vector getProjects(Activity)
Vector getMediaSources(Project)
ProjectVersion getLastModifiedVersion(Project)
Vector getPublished Versions(Project)
ProjectVersion getLastPublishedVersion(Project)
void checkUser(User)
void checkAdministrator(Administrator)
void checkSession(Session)
void checkActivity(Activity)
void checkTeam(Team)
void checkProject(Project)
void checkVersion(Project, ProjectVersion)
void checkMediaSource(MediaSource)
boolean checkInResourceFolder(MediaSource)
void addToResourceFolder(MediaResource)
void removeFromResourceFolder(MediaResource)
Vector getMediaResources(int)
Vector findMediaResources(String, int)

B.2.2 AdminDatabaseAccess

class AdminDatabaseAccess extends DatabaseAccess {
  Overview: The AdminDatabaseAccess abstraction is an extension of the DatabaseAccess abstraction that provides Administration Servers with an easy-to-use interface for updating user and team information.

  AdminDatabaseAccess(String)
  int addActivity(Activity, Administrator)
  void updateActivity(Activity)
  void removeActivity(Activity)
  int addTeam(Team, Activity)
  void updateTeam(Team)
  void removeTeam(Team)
  void addTeamMembers(Object[], Team)
  void removeTeamMember(User, Team)
  User getUsers(Administrator)[]
  int addUser(User, Administrator)
  void removeUser(User)
  Administrator getAdministrator(String)
  Session loginAdministrator(String, String)
  void logoutAdministrator(Session)
  String getPassword(Administrator)
}

B.3 Network connection abstractions

B.3.1 ClientConnection

interface ClientConnection {
  Overview: The ClientConnection interface represents a Crossroads server's network connection to a particular client machine. Using this connection, the server can perform operations such as downloading media sources from the client machine.

  byte getBytes(MediaSource)[]
  String getChars(MediaSource)
}
B.3.2 ServerConnection

interface ServerConnection {
    
    Overview: The ServerConnection interface is the client's parallel of the ClientConnection interface. It represents a client machine's network connection to a particular Standard Server. Client tools can use this connection to perform all their necessary operations that involve a server lookup or update, e.g., retrieving and saving projects.

    Session loginUser(String, String)
    void logoutUser(Session)
    Vector getAdministrators()
    Administrator getAdministrator(Activity)
    Activity getActivity(String, Administrator)
    Team getTeam(String, Activity)
    User getUser(String)
    Activity getActivities(User[])
    Activity getActivities(Administrator[])
    Team getTeams(Activity[])
    Team getAnyTeam(User, Activity)
    Team getTeams(User, Activity[])
    Project getAnyProject(User, Activity)
    Project getProjects(User, Activity[])
    User getTeamMembers(Team[])
    Vector getProjects(Activity)
    Project getAnyProject(Team)
    Project getProjects(Team[])
    VersionInfo getVersionInfo(Project)
    Vector getPublishableProjects(Activity)
    ProjectVersion getLastModifiedVersion(Project)
    ProjectVersion getLastPublishedVersion(Project)
    Vector getPublishedVersions(Project)
    void publishVersion(Project, ProjectVersion, boolean)
    int saveProject(Project, ProjectVersion, MediaContent, Team, Session[])
    int saveMediaSource(MediaSource)
    MediaContent getMediaContent(Project, ProjectVersion)
    byte getData(MediaSource[])
    int getVersionCount(Project)
    Vector getMediaSources(Project)
    Vector findProjectsByTitle(String)
    Vector findProjectsByActivity(String)
    Vector findProjectsByUser(String)
    void close()
    boolean checkInResourceFolder(MediaSource)
    void addToResourceFolder(MediaResource)
    Vector getMediaResources(int)
    Vector findMediaResources(String, int)
    void acquireLock(Project, Session)
    void releaseLock(Project)
    Project getProject(String, Team)
}
B.3.3 AdminServerConnection

interface AdminServerConnection extends ServerConnection {
    
    Overview: The AdminServerConnection interface is an extension to the ServerConnection interface that includes server operations required solely by the Administrator tool.

    int addActivity(Activity, Administrator)
    void updateActivity(Activity)
    void removeActivity(Activity)
    int addTeam(Team, Activity)
    void updateTeam(Team)
    void removeTeam(Team)
    void addTeamMembers(Object[], Team)
    void removeTeamMember(User, Team)
    Session loginAdministrator(String, String)
    void logoutAdministrator(Session)
    int addUser(User, Administrator)
    void removeUser(User)
    User getUsers(Administrator)[]
    Administrator getAdministrator(String)
    int updateUsers(Vector)[]
    void updateActivities(Vector, Administrator)
    void updateTeams(Vector, Activity)
    void updateTeamMembers(Vector, Team)
}

B.4 Content presentation abstractions

B.4.1 MediaPanel

class MediaPanel {
    
    Overview: The MediaPanel abstraction is the primary mechanism for viewing project content, regardless of whether the content is editable or not. Specifically, this abstraction represents a panel in which all project content is displayed. Media sources added to this panel are expanded to MediaComponent objects, which have embedded in them contextual information about the object referencing the media source that can be updated by a user, e.g., location, size, etc. Once added, media sources cannot be retrieved from the panel. However, the MediaComponent objects that rely on the source can be removed from the panel. In an editable state, the panel allows standard editing operations such as ‘undo’, ‘redo’, ‘clear’, ‘cut’,
and 'paste', and other operations such as 'copy' and 'select all,' regardless of whether or not it is editable. For more information on the MediaComponent abstraction, see B.4.1.

```java
MediaPanel()
MediaPanel(boolean)
void setEditable(boolean)
boolean isEditable()
MediaComponent add(MediaSource)
MediaComponent add(MediaSource, int)
MediaComponent add(MediaSource, Properties)
MediaComponent add(MediaSource, Properties, int)
void delete(MediaComponent)
void delete(int)
void clearAll()
void clearSelection()
void set(MediaContent)
void edit(Preferences, int)
void editSelection(Preferences)
void editSelection(MediaSource, Preferences)
void cut()
void copy()
void paste()
void selectAll()
void deselect(MediaComponent)
void deselectAll()
void registerModifications()
boolean isContentModified()
MediaComponent getMediaComponent(int)
MediaComponent getMediaComponentAt(int, int)
MediaComponent getMediaComponentAt(Point)
int getMediaComponentCount()
MediaComponent getSelectedComponent()
MediaComponent getSelectedComponents()
MediaComponent getSelectedComponents(Preferences)
MediaComponent getAllComponents()
```
Component getDisplay()
void destroy()
void setProperty(String, String)
String getProperty(String)
String getProperty(String, String)
Properties getProperties()
void setDisplayX(int)
void setDisplayY(int)
void setDisplayWidth(int)
void setDisplayHeight(int)
int getDisplayX()
int getDisplayY()
int getDisplayWidth()
int getDisplayHeight()