9)

A Distributed Environment for Developing, Teaching, and Learning Algorithmic Concepts

by

Nicholas J. Tornow

Submitted to the Department of Electrical Engineering and Computer Science 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
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 22, 1998

Author _____
Department of Electrical Engineering and Computer Science
May 22, 1998

Certified by _____
Seth Teller
Associate Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

A Distributed Environment for Developing, Teaching, and Learning Algorithmic Concepts

by

Nicholas J. Tornow

Submitted to the
Department of Electrical Engineering and Computer Science on

May 22, 1998

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

Fuse-N is a distributed platform-independent educational environment specializing in algorithmic concepts. The platform is extensible, Web-based, and graphical. Fuse-N provides tools for carrying out administrative tasks involved in teaching such as distributing, supporting, and grading assignments. Developing new educational components is simple, and the platform provides intuitive mechanisms for linking and expanding upon existing components. Since the functionality of Fuse-N is manifold, in this paper we will explore it in terms of three levels of increasing abstraction. At the first level are specific algorithmic concepts. At the second level are user-specified compositions of those concepts. At the third level are the inter-user interactions that take place throughout Fuse-N. Emphasis will be on the interface and exposed functionality of Fuse-N. Technical and implementation details are covered in an appendix.

Although this thesis discusses many aspects of Fuse-N, it focuses on the development of a set of contextual interaction tools in the third level of the system. These tools allow students and instructors to remotely share portions of their respective environments with others. Sharing can take place in several modes -- from passive to active.

Thesis Supervisor: Seth Teller
Title: Associate Professor of Computer Science

Acknowledgements

The culmination of my work is bittersweet. I have immensely enjoyed working on Fuse-N and have trouble tearing myself away, but I am also glad to complete my work. Many people helped.

Nathan Boyd introduced me to the project and helped me ramp up through remote correspondence last summer. The more I delved into the inner workings of the system, the more I appreciated his well thought-out design and attention to detail. Brad Porter was a great partner to have in the Fall in helping to make Fuse-N what it is today. His work on building the data flow aspects of Fuse-N really took the platform to the next level.

Professor Seth Teller has been a source of inspiration and vision throughout my work here. Whenever I start to get lost in the fine points, I can count on Seth to help me step back and see the big picture. Intel and Microsoft Corporation donated the hardware and software that makes Fuse-N possible. Without their help this research would not have been possible.

My former roommate, good friend, and new Fuse-N designer Aaron Boyd is the first developer to first approach the system as a student in a class. His perspective will help us chart a promising path for the future. Finally, Sophomore UROPs Randy Graebner and Bhuvana Kulkani have been great teammates who ensure the project's continued success. I thank my family and the brothers in my fraternity for their support, and the stars for providing me the opportunity to do this kind of work. It has truly been my privilege.

Contents

Chapter 1: Introduction

1.1 Overview

Fuse-N is a collaborative Web-based educational environment designed for developing, teaching, and learning algorithmic concepts. The system is platform-independent and extensible. It's functionality will be presented using three levels of increasing abstraction.

The first and lowest level of focus consists of individual algorithmic concepts. At this level, full attention is given to a specific algorithmic concept. At the second level, multiple concepts are grouped into data flows. Here, individual algorithms' outputs are connected to the inputs of other algorithms, producing more complex behavior from the input of the first algorithm to the output of the last algorithm. On the third and highest level, people use interaction tools to understand and progress through the lower levels. Fuse-N implicitly supports each of these levels and allows easy maneuvering among them.

Another way of looking at the functionality of Fuse-N involves the use of perspectives. Some people work to develop and extend Fuse-N; others use the system to teach; and still others use it to learn. By moving through the space defined by this set of interactions, we can better illustrate the multi-faceted nature of the platform.

Fuse-N is a practical educational tool. At all stages in its development, the platform has been driven by pragmatic problems in the educational process of algorithmic concepts. Fuse-N is motivated by an attempt to improve this process and has been successfully deployed[1]. Through deployment, we have gathered feedback from students. This feedback has enabled us to focus on the most critical shortcomings of the system and helped guide our development efforts.

There are many theories about effective educational methodologies. But according to Laurillard there are five stages common to all descriptions of the learning processes [Lau93]. These stages are learning structure, integrating parts, acting on the world, using feedback, and reflecting on the rest of the process.

1.1.1 Learning Structure

In an attempt to apprehend structure in a topic, students search for order in the subject material. This step is what separates memorization from more advanced learning techniques. In memorization, knowledge exists as a jumble of unrelated words and phrases. Since there is no ordered way to represent this data, one is forced to devote time and energy into committing each piece to memory. While sometimes such techniques are necessary and valuable, they are often time-consuming and in general fail to provide an understanding of the conceptual underpinnings of the topic material. Learning structure is a way to minimize memorization. In the learning process, students try to comprehend the structure of the material. In doing so, they attempt to tie pieces of data together to form

---

[1] In the Spring of 1997, Fuse-N was used in the Introduction to Computer Graphics class at MIT. Results of a survey given out online were quite positive [Por98].

larger coherent sets from which they can extrapolate to apprehend even more complex concepts.

1.1.2 Integrating Parts

People learn many distinct concepts in any given lecture or other educational experience. Taken individually, these concepts are nearly useless. Only through integrating concepts, uniting parts, can students gain appreciation for a given topic. Thus, when entering a new domain of knowledge, students are prone to feel that concepts are useless, pointless, or misguided. Only after they have developed an understanding for the interplay between many factors do they begin to appreciate the issues and problems of a domain. Once this appreciation develops, students can quickly integrate new concepts to enhance their understanding of a topic. One way to present many aspects of a topic is through the use of different media. Lectures, tutorials, readings, and assignments, when combined, give the student a blend of several modes of interaction with new ideas. While each of observing behavior, listening to speeches, reading, and physical or mental experimentation may alone be sufficient for learning, when all of these media are able to resonate with one another, the student may achieve richer understanding at an accelerated pace.

1.1.3 Acting on the World

Learning by doing is a tried and true technique. When students are given the equipment necessary to do a task and some guidance, they may be able to solve the problem inherent in a learning situation on their own. Such a feat not only serves to justify the problem's solution to the students, but also to improve their confidence in their

own abilities. Curricula that provide no "hands-on" experience miss an opportunity to enhance the learning process. In essence, by providing tools for students to act on the world, instructors tap students' built-in learning tools. Instead of having to rely entirely on text and spoken word, students can develop their own unique internal representation for the topic at hand. An added benefit of this technique is that whereas lectures and texts are often limited in scope, experimentation can be quite open-ended. Students may explore the topic until they are satisfied, and even produce new information beyond the material in the text and the knowledge of the instructors.

1.1.4 Feedback

Without feedback from instructors, students may have trouble knowing if they are making progress. By commenting on students' behavior, instructors focus students' energies where they are most needed. Feedback can also serve as a motivational force to help students who question the quality of their work.

1.1.5 Reflection

Finally, after a topic has been mastered, reflection allows students and instructors to review and potentially improve upon their existing methodologies. Similar to the integrating parts step in some respects, reflection allows students to step back and draw general conclusions about the rest of the learning process. Reflection functions at a slightly more abstract layer, however - one in which the knowledge just acquired may be tied into other areas of expertise to generate new ideas and motivate further exploration.

While these stages of learning may at first seem quite intuitive, many learning environments fail to guide students through all of the stages to ensure true understanding.

Fuse-N explicitly supports these five stages of learning. The system is meant to function in conjunction with, not in place of, traditional educational practices. To comprehend structure, after lectures and reading, students can enter Fuse-N and experiment with properly implemented algorithms to understand their structure; to integrate parts, they can read descriptions and watch demonstrations to get different perspectives on a problem; and to act on the world, they can implement algorithms and experiment with different possible solutions. Feedback from instructors and writing up final reports provide the final two stages of the learning process. All of these topics will be further addressed in Chapter 3: Three Levels of Fuse-N.

The emphasis of this thesis is on functional aspects and principles of Fuse-N, with particular attention to the system's interaction tools. Technical notes and implementation details for the system are deferred to Appendix A. Up-to-date information about Fuse-N and a publicly accessible demonstration can be found through the MIT Lab for Computer Science Graphics Group Web page at http://graphics.lcs.mit.edu/.

1.2 Outline

Structurally, this thesis rests upon a partition of Fuse-N into three distinct layers. In Chapter 1, after discussing motivations and related work, we will give a brief sketch of the platform's history. This background will lead into a more in-depth examination of Fuse-N.

Chapter 2 presents a sample scenario in which students and instructors interact with the system and teach other in order to progress though an assignment. This scenario

will provide a number of interesting perspectives on Fuse-N and will indirectly introduce some of the key features of the platform.

After the scenario sketches out some perspectives, Chapter 3 will progress to a more rigorous description of Fuse-N's functionality through the use of the three layers discussed above.

Chapter 4 focuses on Fuse-N's contextual communication tools. The newest major development in the Fuse-N platform, these tools allow students and instructors to share aspects of their Fuse-N experience remotely.

Once we have made this comprehensive review of Fuse-N's functionality and interface, Chapter 5 will develop a perspective-oriented explanation of the platform and draw parallels between the different roles of those who interact with Fuse-N and the platform's levels.

Finally, in Chapter 6, we will discuss current work, future directions, and draw some generalizations about Fuse-N.

1.2 Motivation

Developing, teaching, and learning algorithmic concepts are all complex tasks. Although in practice people may switch between these tasks to such an extent that their role is difficult to define, in Fuse-N we will assume that tasks are accomplished by people acting in one of three distinct roles: the developer, the instructor, and the student. Developers extend Fuse-N's learning tools; instructors oversee and administer the students' learning; students interact with the learning tools, their instructors, and each

other in an attempt to gain improved understanding of the subject material. Since each role has its own set of challenges and goals, we will examine the issues facing each role individually. Although we will focus on educational roles within the domain of algorithmic development, some of the ideas may hold in more general learning environments as well.

1.2.1 The Developer

Educational content developers must precisely define problems for students to solve, and abstract away irrelevant sub-problems. This is not a trivial task by any means [Rob96]. If a problem is vague or provides no bounds on how accurate the answer must be, or what methods should be used, the problem may quickly become too difficult for inexperienced students. Wary of these dangers, an algorithmic concept curriculum developer's task is first to define an algorithmic concept and then to provide some level of base functionality to the student. If the algorithm is Bresenham's line drawing algorithm, for instance, the developer will first want to present to the student the conceptual underpinning of the algorithm. In this case, the algorithm is one of many algorithms that can be used to determine which pixels should be lit on a rasterized screen in order to best represent a line between two points.

The developer may then choose to abstract away some of the irrelevant details to allow the student to focus on the algorithm at hand. These details include user interface issues of presenting an area in which to work, accepting endpoint pair inputs, and displaying pixel outputs large enough for easy detection of off-by-one errors. One way to do this would be to provide the student with a window sensitive to inputs and capable of

drawing outputs on a large grid. This window could have an exposed function setPixel( ) that when called with two integer arguments and a color, lights a pixel on the screen given by the integer arguments in the specified color. By doing this, the developer abstracts away distracting details and allows the student to focus on Bresenham's algorithm. The student may now concentrate on writing the piece of code that decides which setPixel() calls to make to represent given an endpoint pair input. He or she will not have to worry about display, capturing inputs, feeding them to the algorithm, or piping the outputs back to the screen. The developer has abstracted these issues away in order to simplify the student's learning task. The result is a challenge that is much more reasonable to expect a student unfamiliar with computer graphics to be able to solve.

1.2.2 The Instructor

Instructors of classes involving algorithms choose the algorithms they will cover during the course of the term. They also create an ordered path through these algorithms. In order to simplify the curriculum, algorithmic concepts may be grouped into sets based upon some categorization scheme. These sets may be ordered into a specific sequence to allow later work to build upon work done earlier in the term. In creating coherent curricula, an instructor helps students make incremental progress towards full knowledge of the concepts involved. Instructors also often act as course administrators. They must not only teach algorithms, but also assign work in a timely manner, grade these assignments, keep track of students' progress, and guide them along in times of difficulty. If a group of instructors are working together to teach students (perhaps a professor and several teaching assistants), then the instructors will need to coordinate their efforts and

share information about exceptional or troubled students. Thus, whereas developers focus on creating educational material, instructors handle the manifold tasks of creating curricula, teaching, and administration. While this distinction may be blurred in practice, it makes the explanation of Fuse-N's functionality and interface more cogent, as will become clear in the following sections.

1.2.3 The Student

Students need to learn the concepts that instructors select as relevant. In order to learn, students read independently, interact with instructors and each other, and implement algorithms using the tools developers provide. Effective pedagogy provides mechanisms to allow students to engage in each of these activities. Without reading material, students may have difficulty gaining background knowledge; without interaction students may lose interest or get frustrated with problems that others could help them understand; without an opportunity to implement their ideas, students cannot easily exercise their creativity to explore new aspects of the learned material.

Ideally, developers, instructors, and students could focus solely on the difficult challenges their respective roles present. However, substantial setup costs typically impede initial development, teaching, and learning of algorithmic concepts. These costs include administrative issues such as software licensing, deployment, and familiarization. Setup may have to be repeated if the project moves to different platforms, or even just to other computers. If different students work on different platforms and computers, developers and teachers must prepare information for multiple platforms. Even accounting for the difficulties of finding or creating appropriate software for these

platforms, there are additional issues with consistent behavior across platforms.

To illustrate this problem, consider the simple case of student asked to implement a line-drawing algorithm. Once the student learns the algorithmic concept and begins to code, she will still have to become familiar with the current platform's graphics routines before being able to test implementations. Worse still, if different platforms have different behavior, the student may have difficulties determining the correctness of her implementation through comparison with reference examples on another platform.

As a class progresses, the complexity of assignments may escalate. Enabling students to build upon previous work instead of continually starting anew on each assignment allows this progression to take place without placing extreme time demands on the student. Such progressions also contribute to agreement on the class's focus and purpose. Ideally, an educational software platform would scale to allow use of previous work in complex and unrestricted ways as the class progresses.

While working, students may have questions, and instructors may have answers to those questions. Personal interaction plays an important role in learning as instructors help students and students help one another with problems. In the domain of algorithmic concepts, the problems students run into are often only recognizable as errors within a very specific context. Communication tools that can convey this context along with the problem itself will be more effective in problem solving than those that do not. Thus a communication tool should not only allow questioning students to find people who can answer their questions, but should also be able to provide contextual information specific to the problems students often encounter in the process of algorithmic development.

Fuse-N is designed to address these issues. First, it addresses set-up costs by providing an standardized, cross-platform development environment for developing individual topics. Second, it allows for creative and unbounded elaboration of previous topics through an extensible and flexible data flow architecture. Third, it allows for contextual, expressive interaction and collaboration among its users through built-in communication tools.

Web-based tools are often limited by their dependence upon reliable network connections. Fuse-N, although primarily a Web-based tool, may be run locally with degraded functionality if there is a temporary disruption of network connectivity. When network connections are re-established, students may reconnect to the server with no loss of data. For planned, extended periods of loss of connectivity, the software may be run locally. In this mode, all information is stored on the local machine and not on a Fuse-N server, and interaction tools are not available. In the future, we plan to extend the platform's robustness in this area by implementing a mechanism for replicating local data to a Fuse-N server. This capability would allow students to transfer locally saved work to the server for later use on other computers on the Internet.

1.3 Related Work

Fuse-N is, by its very nature, an agglomeration of many different branches of research. It is a platform for education, software development and visualization, and interaction. Each of these areas covers quite a broad range of active research topics. In this section we will briefly review some of the leading systems in each of these fields.

Since Fuse-N is a Web-based platform, we will concentrate on Web-based platforms. It should be noted, however, that there are excellent tools in each of these areas that do not fit into this category.

1.3.1 Educational Platforms

There are several Web-based educational platforms currently in the market. Two of the leaders in this field are WebCT and Hamlet.

The most comprehensive and well-used of these platforms is WebCT [Gol+96]. WebCT provides a rich suite of course development, deployment, and administrative features. Instructors can easily create new courses consisting of readings, assignments, and tests which students can then access with a Web browser. The platform also supports interaction through chatting and email. WebCT is a general tool that uses features like multiple-choice tests to help students interactively learn material. Fuse-N takes a different approach by focusing specifically on algorithmic development. Although Fuse-N's tools in the area of course creation and administration are not nearly as complete as those of WebCT, by focusing on algorithmic concepts, Fuse-N is able to expose more powerful functionality than a general purpose system like WebCT. This functionality includes dynamic compilation and instantiation of Java source code.

The Hamlet project differs markedly from both the WebCT and Fuse-N projects in that it requires students to run special software on the client machine to take advantage of all of its features [Zac+97]. While this requirement makes Hamlet less useful in heterogeneous environments, it gives it the advantage of having local file access. Both WebCT and Fuse-N rely on servers to hold student information for use in later sessions.

Hamlet, on the other hand, stores information locally. Another advantage of the Hamlet system is that it has the ability to call on other programs on the client's machine, such as Matlab or Emacs. Again, such features result in a loss of generality but can be useful in the right environment.

1.3.2 Software Development and Visualization

Fuse-N focuses its educational efforts on providing tools to develop and link algorithms and students. In order to do so, the platform incorporates a development environment and tools for algorithmic visualization. Software development platforms on the Web generally work through use of a client-server architecture. Code is written at the client end and then sent to the server according to some protocol where it is compiled and somehow fed back to the client.

One exception to this architecture can be found in the Jscheme project [Hic97]. Jscheme allows the client to write code in a Scheme-like programming language [Su78]. Since Scheme is an interpreted language, Jscheme can and does do all interpretation on the client machine. The program then outputs the interpreted result to the client with no interaction with the server.

Another Web-based interpreted language platform is webLISP [Mah97]. WebLISP is similar to Jscheme but uses the Lisp language as opposed to the Lisp dialect Scheme.

Fuse-N has support for linking algorithmic concepts through a dynamic visual data flow architecture. This field was pioneered by the ConMan system among others [Hae88]. In ConMan, as in Fuse-N, individual components can be linked together

through a visual interface. Connections can be drawn in a running system and the changes will be handled on the fly. Another seminal data flow architecture can be found in AVS [Ups+89].

More recently, IBM's Data Explorer provides a robust commercial data flow architecture [IBM98]. Fuse-N's data flow system differs from Data Explorer in several key areas. Since Fuse-N is written in Java, it makes use of built-in strongly typed features of the language. Data Explorer, on the other hand, is written in C and makes use of casting to handle data passing between components.

In order to teach algorithms, Fuse-N uses algorithmic visualization tools. Such tools can be found elsewhere in the Brown Algorithmic Simulator and Animator (BALSA) system [Bro+84] at Brown University and in the Zeus system at Digital's Systems Research Center [Bro91]. One area of recent work in this area is Interactive Illustrations, part of the Exploratory project, also at Brown University [Dol97, Try97, BGG98]. The tools provided by BALSA, Zeus, and related projects allow students to interact with properly implemented versions of an algorithm in real-time through user interface interactions. The visualization tools in these systems are limited, however, in their lack of support for student implementations.

A more interactive approach to algorithmic visualization can be found at the Georgia Institute of Technology, where students can dynamically visualize the progress of their own algorithms. This system is based upon a method of incrementally changing a visualization based upon comments in students' code which are interpreted by a tool

called SAMBA [Sta96]. The end result is a sort of animation produced by the POLKA [Sta+92, Law+92] algorithm animation system.

1.3.3 Interaction

The final major aspect of Fuse-N is its interactive tools. This field is experiencing explosive growth with the emergence of the Web. Some notable players include Microsoft's NetMeeting, which allows people to share applications, and ICQ, which supports chatting and instant messaging.

While Fuse-N does support application sharing, chatting, and instant messaging, it also provides interaction modes specific to algorithmic development.

1.4 First Principles

Fuse-N is a complex platform with many compelling aspects. In developing the system, the development team established a set of "first principles" to guide our development efforts. This section briefly reviews the guiding first principles of Fuse-N.

1.4.1 Data Persistence

Fuse-N provides data persistence. All code, data flow linkages, and textual communication are continually logged to the server, which then provides consistent views across multiple sessions. Work done one day will be there the next, and tools allow reviewing that work. In other words, Fuse-N is stateful. It maintains information about people and their actions and allows the recall of that information.

1.4.2 Network and Platform Transparency

Network transparency allows Fuse-N to look and behave the same across platforms, browsers, and physical locations. Since Fuse-N is written entirely in Java, any machine with a Java 1.0 enabled Web browser can access the system by going to the proper URL. Similarly, a Fuse-N server may easily be created on any platform for which there is a Java Run-time Environment (JRE) and an HTTP server. Installation involves copying the Fuse-N directories to the host machine and creating the appropriate links from the HTTP server.

1.4.3 Consistency

Fuse-N's environment is consistent. Developing, teaching, and learning take place in a common laboratory-like environment. Fuse-N uses a standardized framework to support a set of objects that represent concepts in the real world. This framework paves the way for easy extensibility and adaptability and allows easy addition of new features to Fuse-N through the extension of current objects or the creation of new ones.

1.4.4 Event Transparency and Orthogonality

Fuse-N provides event transparency. Input events can originate from local user interface interactions, from a prefabricated test suite on the server, or from remote students or instructors. Thus any event can have scope varying from local to system-wide. Local behavior is identical in all cases, and in highly interactive sharing, Fuse-N does not differentiate between local and remote events.

1.4.5 Evaluation Tools

Fuse-N's statefulness and logging mechanisms generate data on the server, and introspection tools allow students to review this data. Students can analyze current state, review work, see feedback, and make annotations through the Fuse-N interface. Instructors are provided with tools for finding common errors and automatically replying to questions regarding these errors. Thus students and instructors are empowered with the ability to browse through the platform's information.

1.4.6 Visual Completeness

Finally, Fuse-N is visually complete; it presents a clear and intuitive interface to students, developers, and instructors. Everything has a visual representation: algorithmic concepts are seen in modules, large projects are represented as data flows, and people are shown as avatars.

These features - persistence, network transparency, consistency, event transparency, introspection tools, and visual completeness - help guide the development of Fuse-N. Persistence establishes trust in the system on one computer by providing statefulness. Network transparency extends this trust to other computers with Web browsers. Consistency provides environmental structure, event transparency allows this structure to span the network, and introspective tools allow the structure to be examined. Finally, visual completeness provides an intuitive interface to Fuse-N.

Using these principles to guide our development, we have tried to chart a clear path for the platform's development. In the next section, we will briefly discuss past development stages.

1.5 Background

The general idea for developing Fuse-N comes from Professor Seth Teller's NSF Career Development Plan [Tel94]. In this paper Teller voiced a clear intention to establish "collaborative interactive techniques to improve pedagogy." Fuse-N is one result of work in this area.

Fuse-N represents an individual concept though an abstraction called a "module." Embedded in a module is a learning tool that helps teach the module's algorithmic behavior. The groundwork for Fuse-N's algorithmic concept abstraction was established in 1996. During the fall of that year and the following spring, three graphics-related modules were developed for Bresenham's line drawing algorithm, Cohen-Sutherland's line clipping algorithm, and the Gupta-Sproull anti-aliasing algorithm [Boy97]. Since then, several other modules have been created and integrated into the system, including the individual component modules of a full graphic rendering pipeline (Figure 8).

Modules are linked together into arbitrarily complex data flows using Fuse-N's data flow architecture. This architecture works through an abstraction called the "Concept Graph." Although the framework for the full Concept Graph was crafted by the spring of 1997 [Boy97], it wasn't until fall that the full architecture for linking modules together was completed [Por98].

The third essential aspect of the Fuse-N system, interaction and collaboration tools, provides necessary communication channels. Such tools have been integrated into Fuse-N since its inception, but it wasn't until the spring of 1998 that these tools were optimized for dealing with the newly expanded Concept Graph data flow architecture.

In the fall of 1997, an early version of Fuse-N was deployed to an introductory computer graphics course at MIT. The results from that experience were quite positive [Por98]. Since that time the platform has grown substantially.

Chapter 2: A Scenario

In this chapter, we will explore a sample interaction scenario with Fuse-N involving a fictitious student named Oliver, and his friend Maria. The scenario also involves a teaching assistant (TA): Thomas. Oliver and Maria are new to Fuse-N, but Thomas has become familiar with it. The scenario is contrived, but the interactions are meant to be typical of the platform. In this section we hope to illustrate the features of Fuse-N from a pragmatic viewpoint: that of the people who interact with it.

2.1 The Scenario

Oliver, a student in Introductory Computer Graphics, is given a URL in class at which he will find the first assignment [Ber97]. He uses a Web browser to reach the URL and once there is asked whether he would like to login or create a new account. Since Oliver is new to Fuse-N, he chooses to create a new account and enters in a username and password. Fuse-N checks this information against a list of enrolled students and since it checks out, Oliver is admitted into the system.

Oliver is presented with a large black grid and a popup menu asking whether he would like to get a demonstration of Fuse-N or begin working. Oliver requests a demonstration. At the start of the demo, a cursor moves over the screen and makes several boxes appear on the black grid of the Concept Graph. Accompanying these

movements, a text feed describes what is happening in the system. At first, it says, "To load a module, right click on the grid and select 'Load Module' from the popup menu." Then, as a dialog appears on the screen, the text reads, "A module selection dialog will appear." These textual helpers continue throughout the demonstration.

After the module is loaded, it is opened by right clicking and selecting "open" from the popup menu. Now the black grid disappears and is replaced by a code window and learning tool for the selected module. This module is Bresenham's line drawing algorithm. The demonstration shows a sample interaction with the learning tool. The cursor moves over the learning tool and draws lines demonstrating the use of the tool in several modes. After typing in some code, the demonstration submits the module implementation for grading by clicking the appropriate button. A notification dialog reports that the submission was successfully received.

Now the demonstration returns to the Concept Graph, opens several other modules, and links them together. The resultant data flow is activated by a left mouse click on the root module of the data flow "tree". This click causes data to propagate through the data flow and eventually results in a rendered triangle in the final module. The data flow is the graphics rendering pipeline.

The demonstration ends and Oliver clicks to get his assignment. After looking it over, he gets to work. Just as he starts coding his first module, Oliver sees his friend, Maria, log in. He clicks on her avatar to send her a message. After exchanging greetings, Oliver and Maria work on their respective projects. Oliver soon runs into a problem in one of his modules, however, and after some unsuccessful debugging

attempts, decides to ask for help. He notices a TA, Thomas, is logged in and decides to send a message to him about the problem. Thomas responds that they should share desktops. A moment later, Oliver is presented with a dialog stating, "TA Thomas would like to share your session. Do you wish to allow it?" Oliver clicks on "Yes" and then watches as Thomas's mouse cursor moves around in Oliver's session window. Thomas examines Oliver's code and runs test cases in the module's diagnostic area to pinpoint the problem. Once the problem is isolated and discussed, Thomas disconnects and Oliver moves on with his work.

Oliver then receives a question from Maria about the assignment. He clicks on her avatar and asks her if she would like to share an interaction module. Maria responds affirmatively, so Oliver opens an interaction module then clicks on it and selects "Sharing..." He chooses to share the module with Maria, which causes her to receive a dialog asking for confirmation. She is again affirmative, so Fuse-N brings up an interaction module on both machines. This module consists of an interactive whiteboard, a chat system and optionally video and audio conferencing. Oliver and Maria begin to converse. While conversing, they continue to work in their own respective projects. Eventually, with Oliver's help, Maria solves her problem and the shared module is closed.

After making some progress, Oliver realizes he needs a matrix multiplication module, and recalls that this module has been provided by the course administrators. He also remembers hearing from Maria that she had personally made and published a better multiplier module the day before. Oliver decides to open both modules to compare their

performance. He looks in the class directory and finds the module the instructors provided. Then he looks in Maria's public directory and finds a module that she chose to publish there called "improved-matrix-multiplier". He opens them both and compares their performance by timing some complex matrix multiplication runs. Indeed, Maria's module is faster, so he chooses to use that one for use in his project.

Once he thinks he has all of the modules in the pipeline working properly, Oliver tries to render a simple pyramid. It doesn't quite come out right, so he switches to debug mode. In this mode, when data is transmitted along links in the data flow, the links light up. The flow can be stopped at any point and the time-stamped logs of the data that has flown through the links examined for correctness. Through this examination, Oliver realizes that his Bresenham line-drawing algorithm has an off-by-one error. He therefore turns off debugging mode and opens the module to examine its behavior on boundary cases. By opening the module, Oliver gains the ability to run test cases on it without affecting any modules downstream in the data flow. He also gets intuitive visual representations of specific input instances and their corresponding outputs.

Once he has a working pipeline, Oliver receives another message from Maria asking if he has figured out the Cohen-Sutherland line-clipping module. He responds affirmatively and Maria asks if she can share that module to test out the rest of her pipeline. Oliver chooses to allow the sharing and marks his Cohen-Sutherland module as sharable with Maria. Then Maria places Oliver's module in her local pipeline and passes her inputs through. These inputs are automatically piped into Oliver's local machine as well since the module is shared, and he looks at the pipeline's final outputs to verify that

Maria's inputs should generate a partially obscured rectangular solid. After some experimentation, Maria realizes that her z-buffering module is not working quite right and decides to fix that before working on her own Cohen-Sutherland line-clipping algorithm.

After some work, Oliver has a working pipeline. He decides to package it up for later use in a larger data flow by choosing the modularize option. This option allows him to specify the specific inputs and outputs of the data flow as a whole and then convert the entire pipeline into a single module that can later be used in a more complex project. He submits the resultant large module for grading by clicking the appropriate button.

TA Thomas sees this submission when a dialog box pops up on his screen. Since no students are having any problems at the moment, he decides to look over Oliver's code. Thomas open Oliver's submitted data flow and briefly scans its structure. He notes that he hasn't seen the improved-matrix-multiplier module before, and opens it to find that it was written by Maria. After some examination, he realizes that Maria has indeed figured out a faster way to complete the matrix multiplication problem. Thomas sends a message out to the TA list through the Fuse-N interface to alert them to Maria's new module. He then continues to look over the code and decides that it is excellent work. He writes out some short comments, gives Oliver an "A" and sends out a grading report. This grading report message is logged on the server for administrative use and a copy is sent to the head TA and to Oliver himself.

Oliver, who is still logged in to Fuse-N helping other students, gets his grade report and rejoices. Eventually, he logs off the system.

When Oliver logs in later in the week, he is presented with the messages other people sent him in his absence. He loads up his previous work and looks over a log of his messages and a high level review of his previous work that was transparently logged to the server.

As finals approach, instructors and course administrators review Oliver's work and look over his grade reports. In addition, they examine annotations specific to his account that provide more information on his performance in the class than may be gathered from grade reports alone.

Chapter 3: Three Levels of Fuse-N

We have just explored a sample interaction with the Fuse-N platform. What went on in that interaction? First, Oliver focused on a few individual modules. Then once these modules were developed, he began to link them together. Throughout this process, Oliver interacted with others in the system. In order to analyze this scenario and Fuse-N's functionality, we will divide Fuse-N into three distinct layers and look at them individually.

3.1 Level 1: Modules and Algorithms

The first and most specific level is the module, or algorithm level. At the module level, focus is on a specific algorithmic concept and its associated learning tool. Modules are created when, after instructors choose which algorithmic concepts they want to teach in a class, module developers design the algorithms' learning tools. These tools form the heart of a module and are used by students to gain understanding of the algorithmic concepts, to implement these concepts, and then finally to verify the correctness of their implementation.

A dominant feature of these tools is an interactive visual representation of a module's inputs and outputs. In the Cohen-Sutherland line clipping algorithm module, for instance, this representation shows a clipping rectangle in a larger field. The student

is presented with four modes of interaction with this visualization area. All modes enable the student to interactively specify inputs and see their corresponding outputs.

The first mode, termed Reference Mode, demonstrates the proper behavior of the given algorithm. In the Cohen-Sutherland learning tool, lines dragged over the visualization area cause the algorithm to clip the line to the box. As the line is moved around by dragging the mouse, new clipping points are computed and displayed in real time. Experimenting in this mode helps students understand what a proper implementation of the algorithm will do.

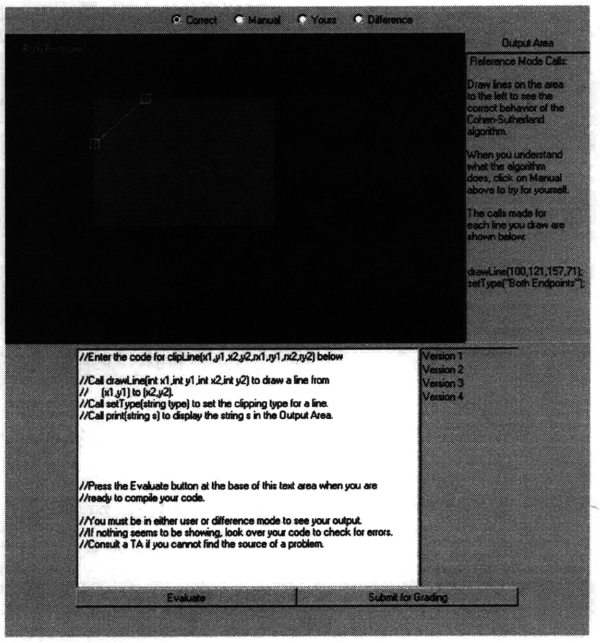

Figure 1. Reference Mode. A student draws a line over a clipping region and the line is automatically clipped.

In Reference mode, as in all modes, students can see the actual methods calls on the visualization area which would have led to the displayed output in a special feedback area. For example, in the Cohen-Sutherland line-clipping module, the base output calls are drawLine (Point, Point) and setClippingType (String). The drawLine call draws a line with the specified endpoints, and the setClippingType call specifies the type of clipping: "Endpoint 1," "Endpoint 2," "Both Endpoints," or "None." In Reference mode, the actual calls made by the reference implementation are shown textually in the feedback area. So one might see the following text on some input: "Reference mode

calls: drawLine (Point (15,24), Point (134, 32)); setClippingType ('Both Endpoints')." This would indicate that the correct clipping points for the drawn line are (15,24) and (134, 32), and that in this case, both endpoints were clipped. An example of interacting with Reference mode is shown in (Figure 1).

After developing an understanding of the correct behavior of an algorithm in Reference mode, a student may choose to move on to Manual mode (Figure 2). This mode challenges the student to manually effect the operation of the reference algorithm though simple user interface interactions. For example, in the Cohen-Sutherland learning tool, a student who draws a line in the visualization area is asked to click on the endpoints of the line clipped to the rectangular clipping area. Feedback from the learning tool helps the student get through this exercise and learn from mistakes. After students have entered a solution, the correct answer is shown to allow the students to evaluate their performance. In Manual mode as in Reference mode, the feedback area supplies method calls as if the student were making function calls instead of clicking in the visualization area.

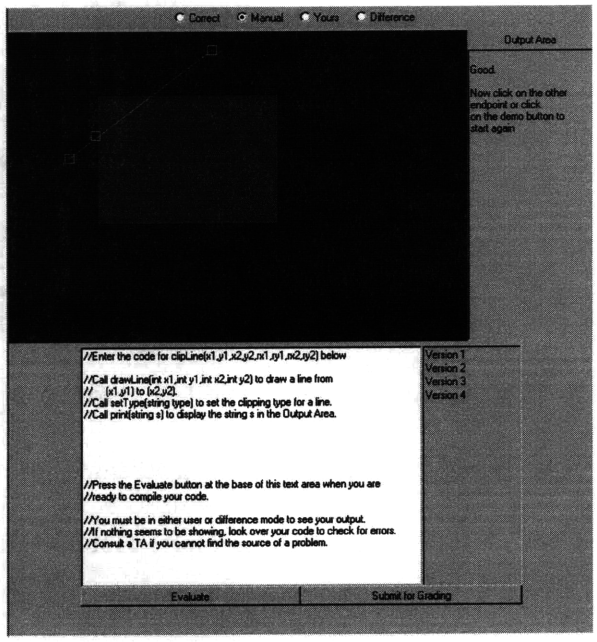

Figure 2. In Manual Mode, the student is challenged to select the end points of the line resulting from clipping a line to a clipping region.

In the third mode, Implement mode, the student gains full algorithmic understanding of a module. Here the student is challenged to write Java code that correctly implements the current algorithmic concept. Entered code can be compiled through the Fuse-N server and dynamically attached to the visualization area. Thus, after a successful compilation, the student can draw lines on the Cohen-Sutherland visualization area and observe the submitted code's behavior on the specified inputs (Figure 3). In the case of an unsuccessful compilation, compiler errors are piped back to the student and automatically appear in a special debugging area (Figure 4).

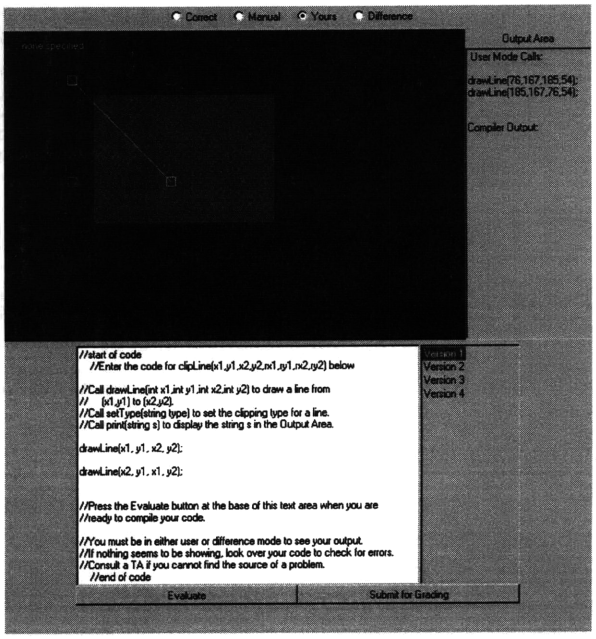

Figure 3. In Implement Mode the student writes code to implement the algorithm. When the student clicks on the "Evaluate" button, the code is compiled and dynamically executed in the visualization area.

Of course, after a failed compilation, the student can choose to modify the code and recompile. Each version of the student's code is logged on the server for later retrieval and students can browse through previous versions of code. If they choose to load an older version, the loaded version will be dynamically bound to the visualization area.

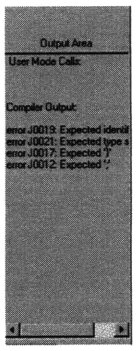

Figure 4. Compiler errors are piped back to the student to aid in debugging of faulty code.

The final module interaction mode, Difference mode, allows visual comparison of the outputs of the reference algorithm and the student's algorithm on identical inputs in the same visualization area. For the Cohen-Sutherland module, the visualization again lets the student rubber band a line between two points. As the student drags an endpoint, the visualization tool draws red squares around the correct clipping points and white squares around the student algorithm's clipped points (Figure 5). By examining the difference in locations of the boxes, the student may be able to gain some understanding about what might be right or wrong about their implementation. When students feel confident in their code, they can quickly verify its correctness by interactively testing out special boundary cases.

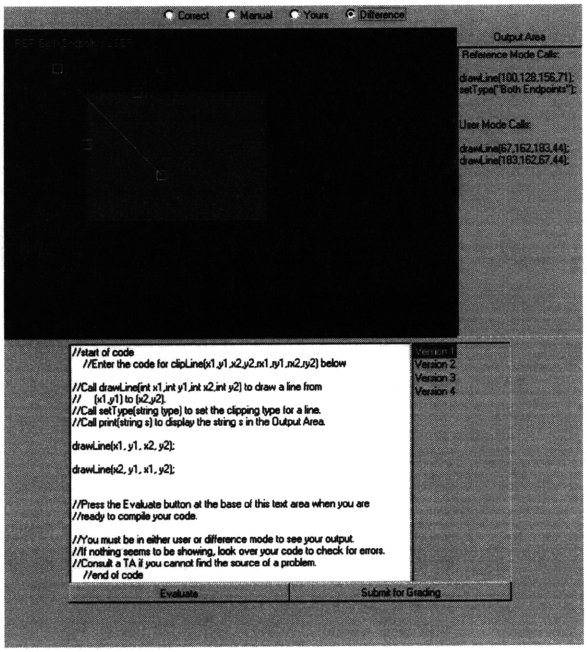

Figure 5. In Difference Mode, students can visually compare their code's output to the reference implementation's output.

Together these four modes: Reference, Manual, Implement, and Difference, help students learn the chosen algorithmic concept, demonstrate conceptual understanding, demonstrate algorithmic understanding, and verify correctness.

When instructors chose to grade a submitted module, they can browse through the versions of a students code, dynamically binding each version to the visualization area.

As new versions are compiled, they are presented to the instructor in real-time. Thus an instructor can watch a student progress though an assignment, or retrace the student's progress by examining intermediate versions of a student's code. This ability, especially when coupled with the interaction tools presented later in this chapter, allows groups of instructors to actively identify trouble spots and help students along with an assignment.

3.2 Level 2: Data Flow and Abstraction

The second layer of Fuse-N's three-layer structure is termed the data flow or abstraction layer. At this layer, an individual algorithmic concept as described above is represented as a rectangular block on a gridded surface called the Concept Graph. Several blocks may be added to this surface and individually moved around using a drag and drop style interface. Associated with most blocks is a set of connectors, representing the inputs and outputs of the algorithm associated with a given block. The Cohen-Sutherland line clipping algorithm mentioned above takes an endpoint pair as input, and outputs an endpoint pair indicating the clipped line segment and a string indicating the type of clipping. Whereas modules are represented as rectangular blocks, module inputs and outputs are triangular connectors. These connectors are bound to the edges of their associated blocks and point in the direction of data flow. Thus, an input connector points into its associated block and an output connector points out of its associated block.

By clicking and dragging from one connector to another of the same type but opposite direction, a connection is established whereby the output of the block attached to the output connector is passed as an input to the block attached to the input connector. A

connection of this sort is represented by a line between the two associated connectors. As one would expect, these connections are maintained as blocks and connectors are moved around in the concept graph. Connectors, and thus blocks, can be connected to zero or more other blocks in this way. Therefore, returning to the module level, we see that at the data flow level, an algorithm's outputs may be linked to another algorithm's inputs to create a complex data flow. This technique of linking one algorithm's outputs to another's inputs allows the student to generate arbitrarily complex data flows.

Blocks, connectors, and connections are three of the main object types of the Concept Graph. Two other object types, the data flow itself, and the avatar will be discussed later in this section. Thus, there are five object types in the Concept Graph (Figure 6). These objects each have an associated set of properties and functionality. The properties can be customized and objects' functionality can be exploited though Fuse-N's interface. Blocks have customizable size, color, position, title, associated connectors, and behavior. Similarly, connectors and connections have their own customizable features.

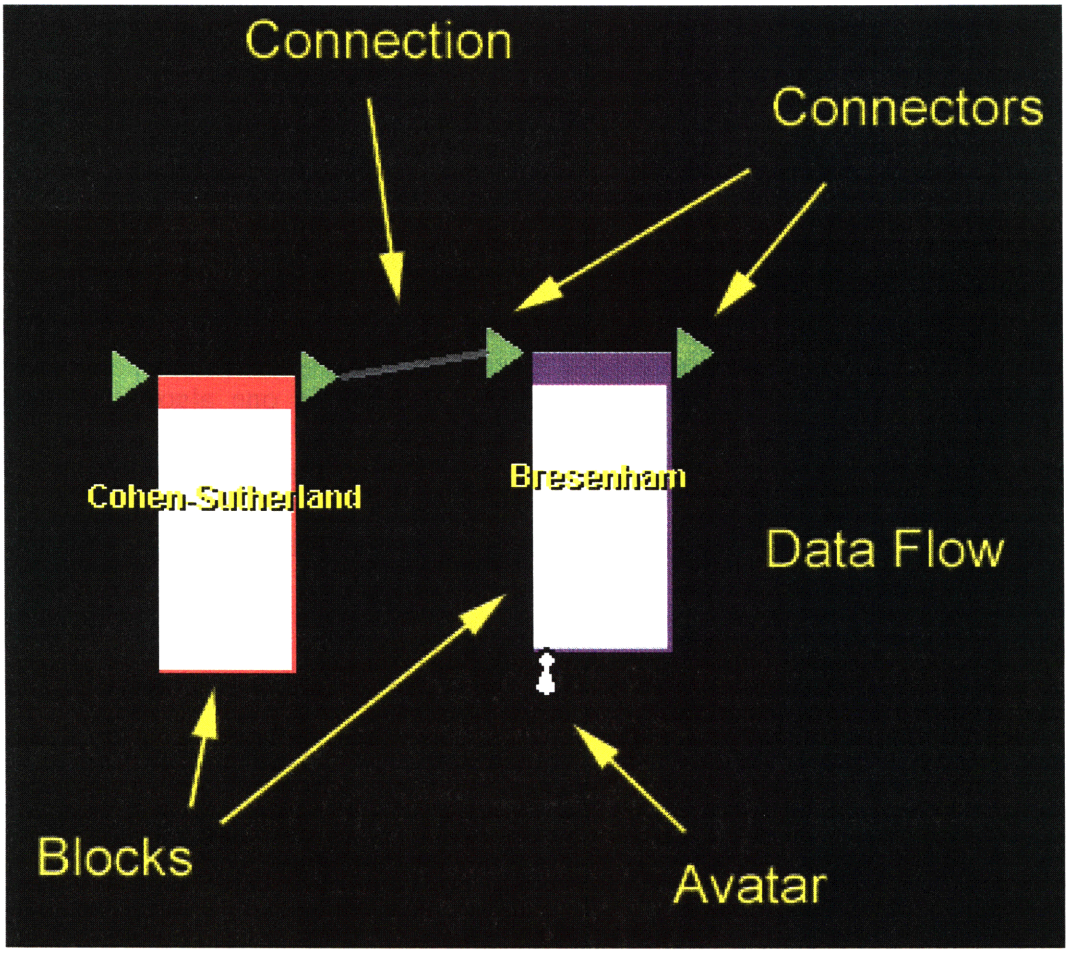

Figure 6. The five major objects of the Concept Graph: avatars, blocks, connections, connectors, and the data flow itself. The data flow object is an object that contains all instances of the other four objects and is represented as the black background upon which the other objects rest.

A block can be customized, opened to reveal its associated teaching tool and code, added to a data flow, removed from a data flow, saved for use in another session, published, or shared. (These last two features will be discussed in the next section) Likewise, the other object types have analogous functionality. Fuse-N is extensible in that properties and functions may be added to objects and new objects may be added.

The five object types currently in the system represent algorithms, their associated inputs and outputs, links between those inputs and outputs, the resultant agglomeration of algorithms, and people.

Data flows are composite objects composed of a set of blocks, connectors, and connections. Like blocks, data flows can be saved and loaded across sessions. If a certain class will be discussing a certain set of modules, a data flow containing the blocks representative of those modules may be set as the default data flow with which all students are presented upon entry to Fuse-N. Students would then be able to open those blocks to interact with the learning tools in the associated modules. Once students have implemented a behavior for a group of modules in Implement mode, they can begin to link the blocks together at the data flow level. If a data flow's behavior is not correct, students can easily drop back to the module level to refine their implementation of a specific algorithmic concept. Their changes will be reflected in the behavior of the data flow when they return to the higher level.

In order to allow Fuse-N to scale when algorithmic tasks cannot be well represented in terms of a few simple algorithms, Fuse-N also supports recursive abstraction. This abstraction technique is implemented by allowing a data flow to be converted into what we will call a "meta-block." A meta-block behaves identically to an ordinary block in the Concept Graph but whereas ordinary blocks reveal code and teaching tools when opened, a meta-block reveals its internal data flow (Figure 7). Thus a meta-block, like an ordinary block, implements a complex algorithmic concept. However, unlike an ordinary block, a meta-block's algorithmic concept is represented in

terms of a data flow of smaller algorithmic concepts rather than as a piece of code. These smaller blocks may be ordinary blocks or other meta-blocks.

Through the use of meta-blocks, Fuse-N allows for unlimited levels of abstraction in its data flow architecture. A meta-block could be opened to show five inner meta-blocks, and each of those meta-blocks could open to five more meta-blocks and so on. This process could continue until eventually a base level was reached at which the blocks were no longer meta-blocks, and would therefore open to reveal not a data flow but instead their own associated code and learning tools.

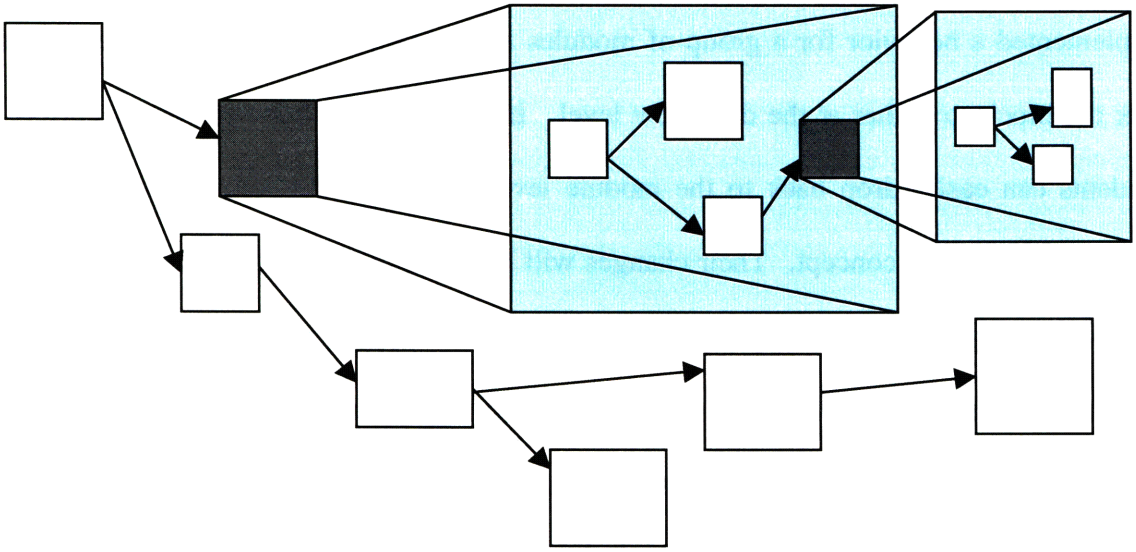

Figure 7. Meta-blocks (shown in gray) function within data flows just like normal blocks, but open to reveal internal data flows rather than code. Meta-blocks can be nested to allow infinite levels of abstraction.

Blocks run as threads in the Concept Graph, and their scheduling is handled by the Java Runtime Environment (JRE) executing Fuse-N. As a autonomous thread, a block may send data out of its all of its out connectors, or any subset of them at any point.

Data received from input connectors may be passed straight to the out connectors, processed in some way that results in output or a change in a block's internal state, or simply discarded. When a block outputs a data structure through one of its connectors, that structure is passed to the appropriate blocks. When data arrives at a block, it is placed onto that block's queue of incoming data; the block's thread then moves through this queue and processes incoming information sequentially.

Since a block is essentially a wrapper and visual representation for a module, a block's input and output connectors generalize to the input and output data types of the block's associated module. When a module is improperly implemented, the block will behave in a correspondingly inappropriate way. Only after several modules have been correctly implemented should the student move on to make complex agglomerations of modules in a data flow. When he or she does, however, the resulting data flow will be nothing more than a linked set of algorithms corresponding to the modules that the blocks involved represent.

The fourth major object of the data flow object system after blocks, connectors, and connections is avatars. Avatars are visual representations of people currently using Fuse-N. A person's avatar appears on everyone's screen as a pawn-shaped figure next to the block the person is currently working on (Figure 8). Thus, if Oliver is working on the Cohen-Sutherland line clipping algorithm, all students and instructors in Fuse-N will see Oliver's avatar adjacent to the Cohen-Sutherland block in their own Concept Graphs. Like all objects in the Concept Graph, avatars have associated properties and functions that can be customized and extended. Many of the features of avatars will be discussed

in the following section.

Avatars bring a new dimension to Fuse-N in that they extend Fuse-N's working environment and community across the network. Whereas most objects can exist in a purely local context, all avatars except that representing the local user are inherently remote. In fact, all Fuse-N objects can operate remotely. This idea is important in understanding the third level of the Fuse-N architecture: the interaction level.

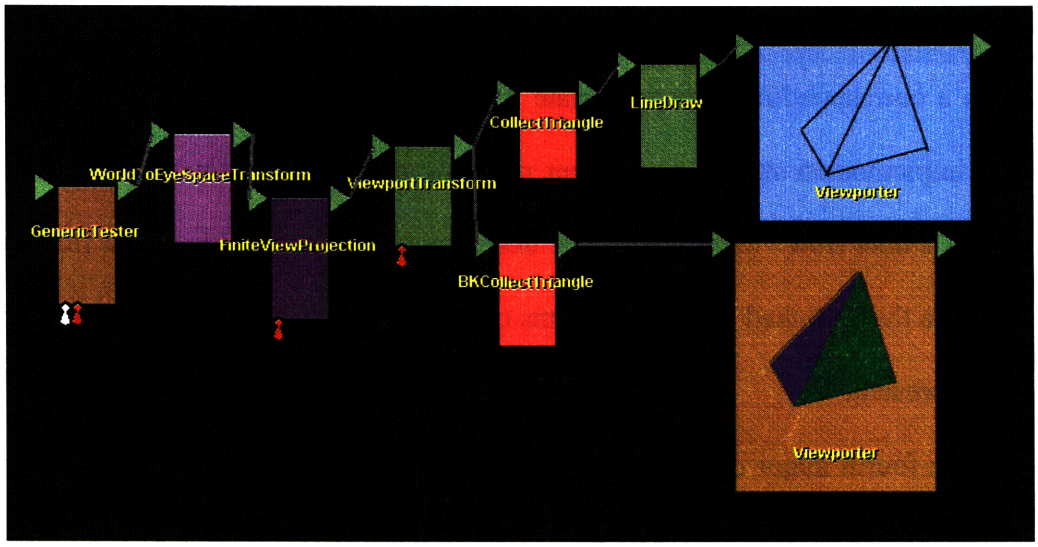

Figure 8. A sample data flow diagram. This data flow shows the canonical graphics pipeline. When the student clicks on the source block, "GenericTester," data flows through the connections to the output blocks on the far right. The outputs of "ViewportTransform" split to two blocks, one of which traces a wire-frame representation of the object, the other of which renders the object with opaque sides.

3.3 Level 3: Interaction and Sharing

The module level and the data flow level encompass all aspects of data and process representation in Fuse-N that occur locally on a student or instructor's machine.

The third level, however, focuses on Fuse-N's Web-based nature. At this level, the interaction level, students and instructors are linked together through a rich set of contextual interaction tools.

There are three basic modes of interaction in Fuse-N. These three modes can be described as being parallel to the three abstraction levels. Thus, there are three types of collaboration in Fuse-N represented by module, data flow, and interaction level interactions.

To enable interaction on the first level, the module level, Fuse-N allows users to "publish" modules. Publishing a module means exposing the executable module (and optionally its associated source code) to other students or instructors. We saw an example of this in the sample scenario when Maria allowed Oliver to load her improved-matrix-multiplier module and incorporate it into his local data flow (Figure 9). By publishing modules, students allow others to load their modules and use them as they see fit. Thus, the act of publishing in Fuse-N is equivalent to changing file permissions to world readable in a network file system. Publishing at the module level is Fuse-N's most basic type of interaction.

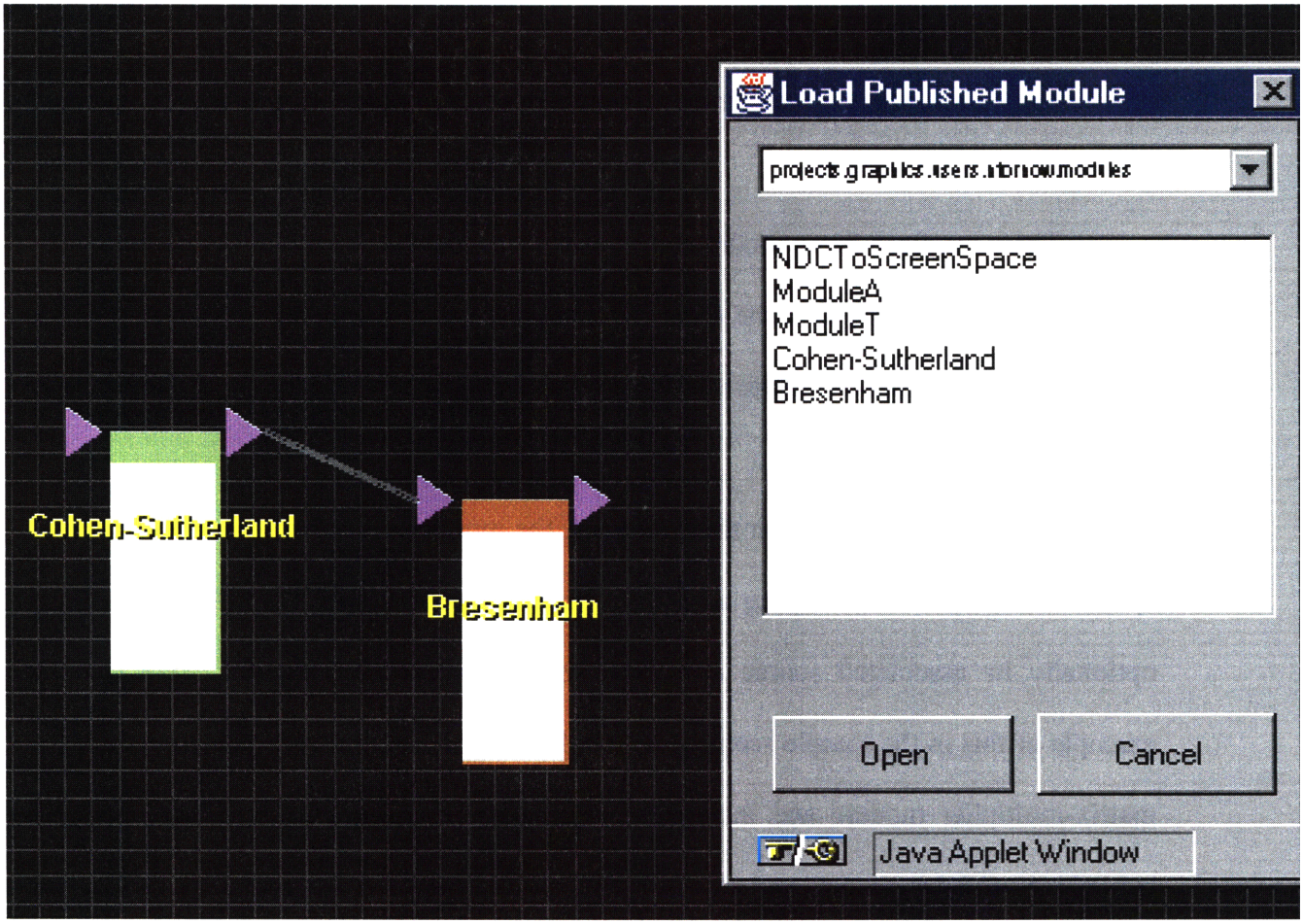

Figure 9. Published modules can be loaded up by others though a file dialog box. Loaded modules function just like normal modules, with the exception their source code may optionally be hidden from others.

The data flow level has its own specific kind of interaction called block sharing. In contrast to module publishing, block sharing is a much more active process. In a shared block scenario, Fuse-N simulates what would happen if there were only one instance of a block among several students or instructors. When two or more people share the same block all input events generated by one person are transparently piped into the input ports of the blocks of those people involved in the sharing process (Figure 10). Although technically each person has their own instance of the block object, sharing

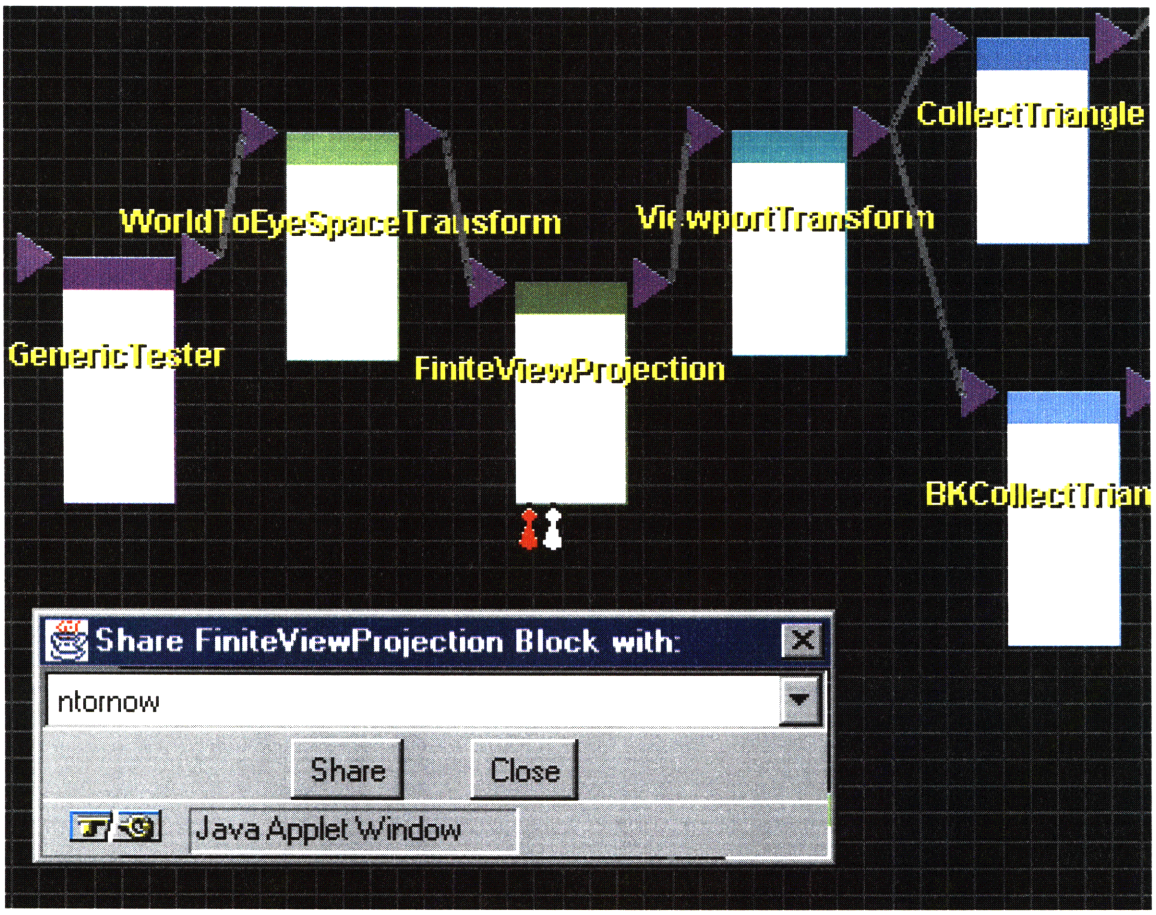

Figure 10. In block sharing, Fuse-N behaves as if there were only one instance of a block among a group of students and instructors.

gives the impression that there is only one instance being shared by all those involved. We saw an example of sharing in the sample scenario when Oliver shared his implementation of Bresenham's line-drawing algorithm with Maria, and again when he shared an interaction block with her. In both cases, the shared blocks received identical inputs, behaved identically, and produced identical outputs on both machines.

At the highest level of interaction, the interaction level itself, a student or instructor may choose to share an entire Fuse-N session with someone else. A session in

Fuse-N is defined as the non-transient state of the client-side applet running in a student or instructor's Web browser. Whereas at the module level and the data flow level, interactions allowed people to concurrently run independent activities within a Fuse-N session, at this level all aspects of a session are shared (Figure 11). In the sample scenario, we saw Oliver use session sharing at one point when he had a bug he could not identify. He decided to share his session with Thomas the TA to help locate the bug. All those involved in the sharing process may interact with Fuse-N as if they were all at a common terminal.

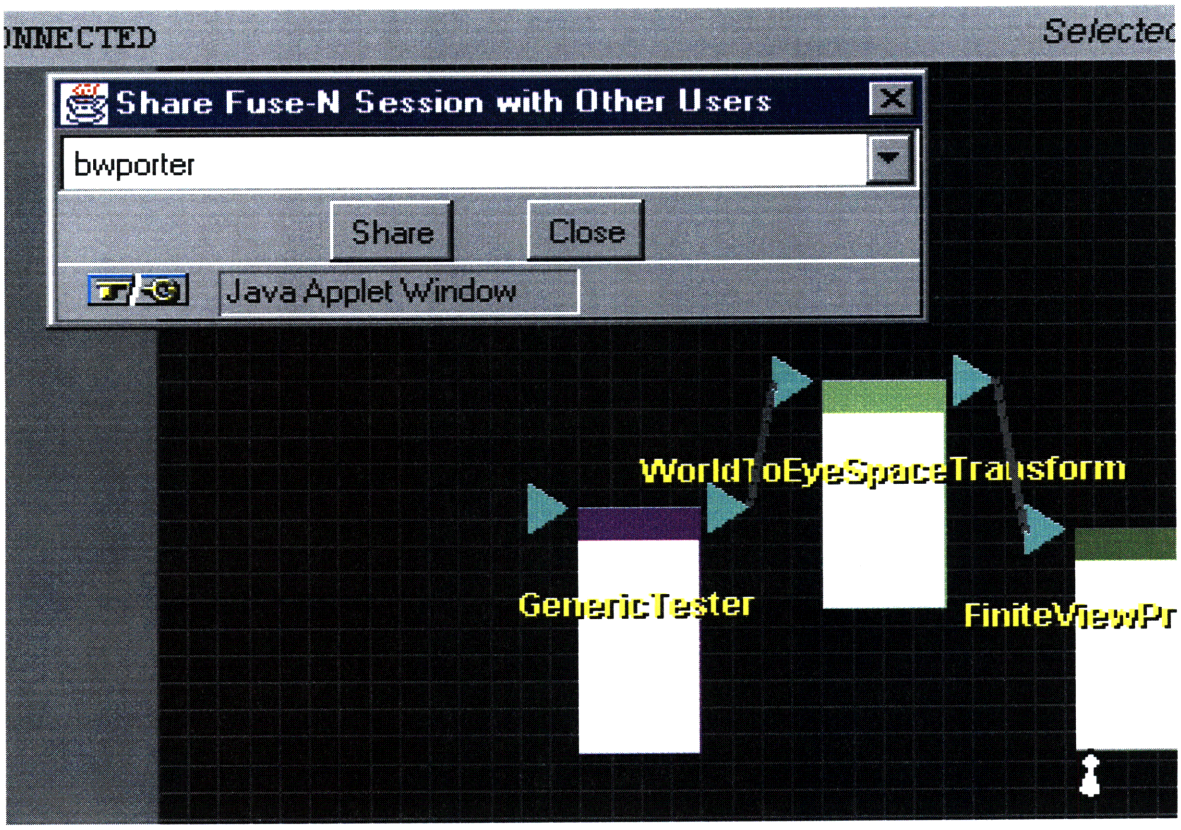

Figure 11. Students and Instructors can easily choose to share sessions. Session sharing allows all parties involved in the sharing to see one Fuse-N session as if they were all gathered around the same terminal.

Obviously, session sharing is a much more involved process than publishing a module or sharing a data flow block. In session sharing, all user interface actions at one console are transparently transmitted to all those involved in the sharing. Thus, if Thomas opens a module block to look at its associated teaching tool and code, Oliver's screen will also change to display the same view. Conversely, if Oliver closes that view Thomas's view will also close. Such in-depth interaction is bandwidth intensive and only useful in certain contexts. One example of such a context might be when two people would like to work together to isolate a bug. Session sharing is also useful when

someone wants to demonstrate an algorithm's behavior on otherwise hard to convey test cases to one or more other people.

Another case where session sharing is particularly useful is in giving prefabricated demonstrations on how to use the system, such as occurred at the start of the example scenario. To produce these demonstrations, actions are saved and later run within the context of the observing student's session.

In addition to these three modes of interaction, Fuse-N also provides tools for chatting, browsing chat history, instant messaging, sharing a whiteboard, reading information posted by instructors, and processing student requests for help in a fair and orderly manner.

Fuse-N's chat system allows for broadcasting and multicasting messages between students and instructors. When messages are sent to people who are not currently logged in, the messages are stored on the server and delivered to the recipient when he or she next enters Fuse-N. Messages are sent instantly through a Zephyr-like interaction system [Coh94]. Students and instructors are presented with a dynamically updated list of people currently logged into Fuse-N. To address their message, they can select one or more names from this list, manually enter in the names of people not logged in, or choose to broadcast to the entire class. When a student or instructor receives a message, a window appears on their screen indicating who the message is from and containing the text of the message. This window can be closed with a simple click. If one wants to look back over past messages, Fuse-N provides tools to do so.

Students may share an interactive whiteboard with one or more other students, or

simply use one locally to sketch out their ideas. Every time someone logs in they are presented with messages from the instructors and some general information about the platform. Help screens provide assistance in case a student becomes confused, and several demonstrations of students' interactions with Fuse-N are available for viewing. When these demonstrations are activated, the student can watch and learn from a pre-defined interaction script and helpful textual narration. If the instructors allow it, students may also record sessions for later viewing. If a student records a session and then makes it world viewable, then other students and instructors will be able to watch a "movie" of the student's session.

If students require the assistance of an instructor, they may enter their name on a help queue. Instructors can look at the queue to determine who has been waiting for help the longest. Once a student has been helped, they can remove their name from the queue to notify other instructors that their problem has been resolved.

Fuse-N also supports the inclusion of more traditional media. Hyper-links to related course material can be integrated into Web pages, and the email addresses of instructors and other students can be referenced through the Fuse-N interface[2].

All messages are logged on the Fuse-N server and can be browsed later. They thus serve as a sort of transcript of a student's textual interactions with other students. To

---

[2] As an aside, when we deployed the system in the Fall of 1997, we used email to provide students with feedback and to notify them of their grades. At this time we planned to give students their grades through Fuse-N's instant messaging system instead of though email. After some discussion, however, we realized that email would be a better method since many students check their mail several times a day and generally only log into Fuse-N when they have specific work to do for a class.

complement this transcript, Fuse-N provides related methods of storing state about a Fuse-N session to students. The system can be configured by instructors to textually log student actions in real-time. This feature allows students to review their past work and teachers to gauge student progress. If more comprehensive logging is needed, the student may record the session as described above to create a re-playable movie.

Chapter 4:

Contextual Communication Tools

4.1 Overview

In workplace and leisure activities, textual, video, and audio demonstrations may be the best means of communicate information. In highly technical domains, however, often there is no substitute for seeing exactly what someone else is doing. In the process of programming, for example, bugs can be extremely difficult to trace down without knowing the complete context in which a piece of code is operating. Traditional Web-based communication tools are extremely efficient in conveying certain types of knowledge but often do little in terms of providing hands-on interaction. Fuse-N places a high degree of emphasis on incorporating the concepts of sharing and collaboration into its infrastructure. We feel this investment in communication is crucial to the learning environment that Fuse-N seeks to create.

In Chapter 3, we talked about the three main levels of Fuse-N: the module level, the data flow level, and the interaction level. The first two levels make up Fuse-N's version of the traditional programming environment. This programming environment is quite flexible considering that Fuse-N's main emphasis is on education and not the

development environment. The final level of Fuse-N, the interaction level, moves Fuse-N beyond traditional programming environments by taking advantage of the Web's real-time interactive potential.

By providing all students and instructors with an identical interface, and allowing those interfaces to be shared, the technical details of solving problems programming complex algorithms is substantially reduced. In effect, the network becomes transparent and instructors and students interact as if they were all in the same room full of identical computers were capable of walking over to other people's screens. Compare that to working on isolated computers with various compilers on different platforms and communicating with email or even videoconferencing. These general interaction modes force information to travel through an intermediary before transmission. Thus if a student is having trouble with a piece of code, she will have to explain her problem in text, speech, gesture, or some combinations of these, and the listener must convert these actions back into a specific situation. Fuse-N's sharing modes cut out the intermediary and allows people to interact directly though Fuse-N's interface. The result is more precise and prompt communication. By placing emphasis on collaboration and interaction, Fuse-N allows the learning process to proceed more efficiently than if students were forced to use external communication tools.

4.2 Details

4.2.1 General Implementation

Publishing modules involves a rather simple process of transferring files to appropriate publicly readable directories. When students chose to load a published module, they are presented with the publicly available directories and allowed to select a module.

The more active modes of interaction, session and module sharing, operate through the use of a specialized server called the Event Server. This server keeps track of everyone that logs into the system and handles the complex real-time issues involved in session and module sharing. The Event Server keeps a hash table of streams to each client machine. When a user chooses to initiate a share, the server adds a new link to the streams associated with each of the clients involved in the share. This link points to the other streams involved in the share. Thus if Oliver and Maria chose to share Bresenham's line drawing module, the Event Server will create tagged links from Oliver to Maria and from Maria to Oliver. The tags on the links will indicate that this is a module share and that the module involved is a specific instance of Bresenham's algorithm. When Oliver's module receives an input, the input will be sent to the server and back out to Maria. Likewise, when Maria's module gets an input, the input will flow to Oliver's module.

If Oliver and Maria had chosen to share a session, the links would be tagged appropriately, and user interface events, rather than more general data types would be routed between users. To avoid network congestion, since the client knows what type of

shares it is currently involved in, it will only send relevant data out to the server. This client-side filtering not only reduces bandwidth usage, but reduces the load on the Event Server.

4.2.2 Module Sharing Issues

When multiple shares are in progress, the server will use the tags associated with individual links to decide where to route incoming data from a client.

When multiple instances of a shared module exist in one of the sessions involved, the issue arises of how to select which instance is shared and which is purely local. Fuse-N deals with this potential issue by using unique identifiers for each instance of a module to handle routing of sharing data. These identifiers are abstracted away from students and instructors.

Another immediate issue we faced in module sharing is what to do when the people involved in a share have different versions of the same module. When one user has a working Bresenham module, for instance, and another other has a broken module, then how can data be shared? The answer we have settled on is to sharpen the definition of module sharing to module *input* sharing. Thus, when modules are shared, they receive the same inputs. Their behavior on these inputs may be different, but their inputs are guaranteed to be identical. Of course if two students want to make their modules behave identically, they should make sure they are both using the same versions of a module. One student could publish their version, for instance, and then allow the other student to load it. The students could then share their identical modules and tie them into their

respective data flows. If they share only that one module, then the rest of their pipelines will behave independently as expected.

One limitation of our current model for module sharing is that all data types that are passed through shared connectors must implement an interface which we call the Persistent interface. The Persistent interface requires objects to have write and read methods that transfer all of their non-transient state across the network using standard primitives. While for most simple data types that a developer creates this is not too difficult, for more complex data structures like hash tables and images, implementing the Persistent interface can be non-trivial.

What we see after some exploration of the problem is that module sharing would be much more flexible if all built in Java data types implemented our Persistent interface, or more generally, provided methods to transfer their non-transient data. The property these methods would provide is often termed object serialization. While object serialization is built into more recent versions of Java, current browsers only support Java 1.0[3]. This lack of support has led us to develop Fuse-N primarily according to 1.0 specifications and we have thus been forced to re-implement some of Java 1.1's features in Java 1.0. Specifically, we have developed methods of serializing storing Concept Graph objects, user interface actions, and the data structures passed around in the data flows we have thus far constructed. These methods allow us to pass events and objects from one student's module through the network to the server and on to other students'

---

[3] Java 1.1 supports many advanced features not included in 1.0. These include object serialization and remote method invocation(RMI) [SM98].

modules. We are limited, however, in that we must write new code for each object that we wish to enable for serialization.

Anticipating the emergence of new Java 1.1 browsers, and realizing the limitations in our current model, we have written code that abstracts away the details of the transmission of an object's state across the network so as to smooth the eventual transition to Java 1.1's implementation of this technology. With full object serialization, Fuse-N's module sharing ability will expand enormously to allow the use of all built in Java data types.

4.2.3 Session Sharing Issues

There are three serious problems that come up in session sharing that do not arise in more passive sharing modes. The first is concurrency. When several students involved in a session sharing simultaneously move the same block in a concept graph, or simultaneously type code, the server must somehow decide who acted first. The chosen action must then be propagated to all of the people sharing the session and undo over-ruled actions. We have experimented with several ways to deal with this issue, and have settled on a system whereby at most one student may "have the floor" at any one time. Students can only interact with their session when they have the floor, and they can only gain control of it when the server grants it to them in response to a request. This system seems to scale well and avoids many of the more complex issues involved in concurrency control.

The second major issue of session sharing concerns dealing with the initial synchronization of several student's sessions. If a student were allowed to start a process of session sharing and then manipulate their session before other students had begun to share, then synchronization could become quite complex. Instead, we have chosen to freeze all students' sessions during a short initial synchronization period. In this period, no one may interact with their session. Once synchronization is achieved, students are allowed to attempt to gain control of the floor and interact with their session. We are currently looking at loosening these restrictions through active logging of session sharing events. With logging, synchronization could happen in real-time; those who fall behind the action are fed a sort of "instant replay" of the events that they missed out on.

The third main issue of session sharing concerns bandwidth. As students move their mouse pointer around on the screen and interact with Fuse-N through other input devices, they generate a tremendous amount of data. If the platform attempts to transfer all of this data to the server and then to all of the students involved in the share, network bandwidth will almost certainly be saturated, and latency will result.

There is hope, however, since productive sharing may still take place when only a fraction of the data is transferred. Suppose a student moves her mouse around on a Concept Graph - starting in the upper right corner of the screen and moving to center where she clicks on a Bresenham module to open its teaching tool. In the process of this simple interaction, she has generated tens and perhaps hundreds of user-interface events corresponding to mouse movements, clicks, and timing. But the interaction could be effectively reduced to two events: a mouse move and a mouse click. By exploiting the

huge reduction in information that this insight provides, we can substantially reduce bandwidth and latency in session sharing.

Fuse-N currently transfers only mouse clicks, mouse drags, and keyboard events; we are exploring dynamic methods for determining subsets of mouse movement events to transfer based upon real-time latency measurements. Such a system would enable those for whom bandwidth is not an issue to see more mouse movements than those on bandwidth-limited systems. Seeing the actual mouse movements of other students would not only increase the immersiveness of the session sharing experience, but would enable more advanced communication. A student may trace a shape with her mouse to convey a concept, or move over areas of another student's code to establish a common focus point among those involved in a share. As mentioned above, however, this added benefit increases network congestion and latency.

4.2.4 Other Interaction Tools

As mentioned in Section 3.3, Fuse-N provides other interaction tools besides publishing, module sharing, and session sharing. Whereas these three tools are highly contextual, other tools like chat, messaging, whiteboard, and related systems in Fuse-N allow students and instructors to interact without inherently establishing context. They are thus more "light-weight" in the sense that they have little or no dependence on the current state of a person's session. Publishing requires that the student produce a module, module sharing requires two or more students to simultaneously use a module, and session sharing requires complete synchronization of several Fuse-N sessions. Light weight, or non-contextual, communication tools, however, place no restrictions on their

users. They thus complement the contextual tools by allowing short and simple interaction with minimal disruption to a student's work. We are actively exploring how these tools can best be woven into Fuse-N's interface to facilitate learning.

Since Fuse-N is a visual environment, one issue concerns how to visual present interaction options. Contextual interaction tools could be associated with a person's avatar, or with the shared object's representation, or both. We are also exploring methods of tying non-contextual interaction into Fuse-N's interface so that the student is aware of incoming communiqué but still able to continue work without becoming distracted.

Communication is often challenging even in face-to-face situations. While we do not plan to provide a complete suite of communication tools to "solve" these difficulties, we are actively attempting to make communication in Fuse-N as transparent and powerful as possible.

Chapter 5: Perspectives

In Chapter 3, we described Fuse-N using three abstraction levels: the module level, the data flow level, and the interaction level. But Fuse-N may also be described in terms of the various perspectives of the different types of people who interact with it. One grouping of the essential roles of these people is the module designer, the instructor, and the student. Interestingly, these roles parallel nicely with the levels described in the previous sections. In this chapter we will briefly outline these parallels, and in so doing, provide another criterion for splitting up Fuse-N's functionality into three distinct levels.

5.1 The Module Designer

The first role, that of the module designer, deals primarily with crafting workable representations and teaching tools for algorithmic concepts. The representation the designer chooses is used to construct the visualization area of a module. The teaching tool works with the chosen visualization to take the student through the four modes of a module: Reference, Manual, Implement, and Difference. Since the designer does not have to worry about how this module will interact with other modules, or how students using this module will communicate, one might say that the module designer works exclusively on the module level.

5.2 The Instructor

The second role, that of a course administrator or instructor, focuses on creating as coherent as possible of a progression through the concepts to be covered in a class. If the class progresses by building upon work done earlier in the term, perhaps after many low-level concepts have been introduced and explored, then algorithmic complexity will tend to increase. To deal with this increase, the instructor will want to abstract away the details of earlier algorithmic tasks. Since such organization tasks are concerned primarily with structuring sets of algorithmic concepts hierarchically or otherwise, one could say that course administrators work at the data flow level. At this level, blocks are built and linked to earlier blocks, and groups of blocks are combined to form meta-blocks that work within a larger framework. Instructors make plan these progressions and help students to understand the rationale of the progression. Certainly, instructors have some role to play in the other levels as well, but the organizational issues are well matched to the data flow level since data flows deal primarily with arrangements of small components into larger structures.

5.3 The Student

Finally, the student interacts with other students and instructors while trying to solve computational tasks in the current assignment. Thus, the student works on all three levels of Fuse-N. This places students on the third level: the interaction level. A chart depicting the structure just described is shown in Figure 10.

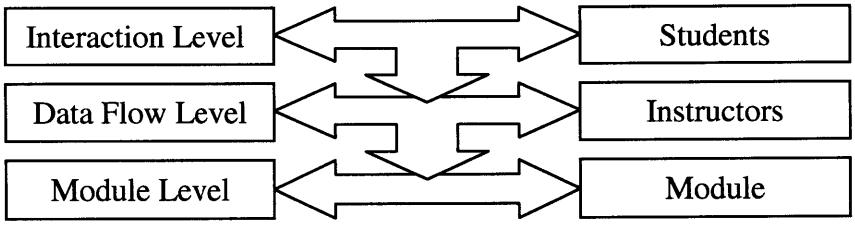

Figure 10. The level abstraction has clear parallels to the different roles people play in interacting with Fuse-N. The downward arrows are meant to indicate that those at the higher levels use their own level as well as those below.

From a technical standpoint, these correlations are loose. Instructors do interact with students and other instructors, and module designers are probably somewhat concerned with linking their module with other modules. The essential point to take from these correlations is that the abstraction techniques of Fuse-N allow module designers and instructors to focus specifically on certain tasks without worrying too much about the others.

Chapter 6: Future Work and Conclusions

6.1 Future Work

Fuse-N's visual completeness entails the use of several distinct abstractions. Primary among these are modules and connections, which represent concepts and conceptual links; data flows, which represents composite projects; and avatars, which represent people. Each of these objects in Fuse-N has a robust set of associated methods and data. So for instance, students and instructors represented in the system through avatars are customizable, and these customizations are persistent.

One large area for potential future work lies in developing powerful tools for browsing information associated with objects and creatively customizing them. Permissions and security are certainly factors that will have to be addressed in the process. By making objects more dynamic and stateful, the Fuse-N environment will become richer. While many features have been built into the platform as it progressed, there are certainly many interesting options for increasing the richness and statefulness of the Fuse-N environment.

For example, modules could be associated not only with Fuse-N learning tools, but also with other sources of information on a given algorithmic concept such as URLs of related sites. More modules properties like shape, documentation, revision history log,

student annotations, and bug reports could make them even more rich and useful.

In general, as Fuse-N grows more complex, the number of parameters which could potentially be customized increases. This customization may include tuning the functionality and user interface of the system. As Fuse-N continues to become more powerful, the ability for customization will continue to grow. We must be sure not to hide all customization features, nor to overwhelm people with an unmanageable set of options.

Editing code is an important part of Fuse-N. The editor should be highly functional. Color highlighting of key words, automatic indenting, and search and replace are all important features of a good editor. We are currently working on integrating such an editor into Fuse-N. This integration process is not trivial, however, for the editor should be tied into other aspects of the system like debugging and module learning tools.

Obviously, debugging tools should be as robust as possible to speed implementation difficulties. These tools must function not only at the module level, but also at the data flow level. We are working on developing a Java-based Java debugger for use in Fuse-N.

Although Fuse-N provides a nice set of administrative tools, these tools could benefit from further development. Sets of testing routines for instructors to quickly verify the correctness of a students code, tools to help instructors react quickly to questions, to spot questions before they are asked, and to allow general course administrative duties such as grade keeping, announcements, and assignment distribution would all be helpful additions to the platform.

Since Fuse-N is based on modules, it is important that these be standardized and easily understood and developed. With this in mind, we would like to generate a rich set of module development tools to allow relative newcomers to Fuse-N to quickly begin adding functionality to the system in the form of new modules

Fuse-N's servers have not yet had problems with load but may in the future as interaction tools push the bandwidth envelope and the user base continues to grow. Thus, it will be important to balance server load to guarantee good performance and allow real-time interactions.

We are currently working on developing modules for use in Structure and Interpretation of Computer Programs (6.001) at MIT in the coming year. This project entails creating a Scheme interpreter module for Fuse-N in Java. The fact that this extension does not pose a serious challenge to the existing architecture of the platform bodes well for continued extensions.

6.2 Conclusions

Although Fuse-N is only in its infancy in terms of its potential, the platform already encompasses a wide range of functionality. After having been successfully tested and then substantially improved, we have high hopes for its performance next year as its usage could potentially more than triple if the platform is used in 6.001.

Fuse-N encompasses all of Laurillard's pedagogical learning stages. In doing so, it aids students in developing an understanding of complex algorithmic concepts.

In this paper, we used a level abstraction of the Fuse-N architecture to discuss its

power. At the module level, the focus is on algorithmic concepts. At the data flow level, disparate concepts are integrated into complex inter-connected data flows. At the interaction level, the focus is on people, problems, and answers.

It is our hope that Fuse-N will continue to succeed in the classroom and will develop into a more powerful, more focused teaching tool. The architecture is robust and extensible and the Web continues to grow. Fuse-N offers a reliable, highly functional cross platform solution to teaching algorithmic concepts on this growing medium. In doing so, it minimizes setup costs, eases developing teaching tools for new algorithms, streamlines course planning and administration, and provides a persistent and consistent interface to students and instructors. The platform is highly integrated with a rich suite of contextual interaction tools to allow quicker communication and thus hopefully less frustration, quicker problem solving, and, in the end, more efficient teaching and learning.

As Fuse-N continues to develop, our focus will continue to be on empowering the designers, instructors, and students to flexibly and exploit the platform's power. The first principles outlined in Section 1.4 chart a clear path of continued growth and we are optimistic for the future.

Appendix A: Implementation

Fuse-N is implemented as a client server architecture. All code on both ends is written entirely in Sun's Java programming language (JDK 1.0) [Gos96]. Since Java applications run on any machine with a Java run-time environment, and Java applets run in most modern Web browsers, Fuse-N's client and server can run on almost any platform.

The Server

On the server side, Fuse-N consists of a Java servlet [SM97] which handles brief, stateless transactions. Among other things, the Java servlet is responsible for compiling code, serving HTML documents, and password checking. Although servlet threads makes changes to the file system on the server and send information to the client-side applets, connections to the servlet are terminated at the end of each transaction and the hosting thread is killed. One advantage of Java servlets over Java servers is that since a new servlet thread is launched automatically by the Web server whenever a request comes in, servlet reliability is essentially equivalent to Web server reliability, which can often be quite good.

In order to deal with longer, stateful transactions, Fuse-N also runs three Java servers. One server handles user tracking within modules; another handles chatting, and

other real-time messaging; the third server handles event sharing and object serialization. Although there is technically no reason to run three servers on the host machine, we found this the easiest method in terms of modular development strategies and reliability factors. By having several Java servers, we are able to take servers down individually to make minor adjustments while the system is in use. During the down time, client-side functionality is degraded, but the system maintains some connection with the server, and of course, the servlet is functional (unless the Web server is also compromised).

The manifold nature of the server side has advantages and disadvantages. On the down side, developers new to the system have some learning to do before they understand which server or servlet is handling individual requests. On the upside, the connections with the client are more robust and can be temporarily compromised without total loss of connection. The division of function among server side components makes incremental improvements easier and improves modularity.

The Client

On the client side, Fuse-N consists of a single applet which opens three continuous connections to the various servers and conducts transactions with the servlet whenever necessary. The applet has several different interesting technical aspects. For one, it is quite simple and simply lays out the components visually, handles some routing between components and initializes some state. The applet contains a set of Panels, a generic Java Advanced Windowing Toolkit (AWT) element for marking off screen space.

The main panels for the client side applet are the Concept Graph panel, the editor panel, the diagnostic tool panel, and the collaboration panel. The Concept Graph panel handles the display of Concept Graph objects: blocks, connectors, connections, data flows, and avatars. The editor panel provides a text entry area and a suite of tools for loading, saving, compiling, and submitting code. The diagnostic tools consist of a visualization area, a selector for moving through modes, and a feedback area. The collaboration panel is highly customizable and is the general area where users browse chat histories, interact on whiteboards, and retrieve grade reports.

The applet is hierarchically defined in terms of layout with the main applet calling in the four main panels, and those panels calling in any sub panels they may have. When possible, we have separated visual elements from event handling elements and both from the central computational elements.

Structure

The code is structured in packages, which are equivalent to directories in a traditional file system. The root directory has six subdirectories that contain the objects within those areas. The six main directories are for the concept graph and its associated objects, the editor, dialog boxes, networking, utilities, and debugging. Thus each of these topics is its own package, and all of the code for dealing with specific aspects of the system is located in the same directory.

Networking

One of the many interesting aspects of the technical side of Fuse-N is the networking system, which was first created by Nathan Boyd [Boy97]. As mentioned earlier, Fuse-N has three Java servers. Each server listens on a separate port. When the first server finds a connection it generates a new 32 bit random number which is the client's unique identification number. In subsequent connections to the other servers, this random number is passed back to the servers, which then start a new thread called a server handler. The server handler adds the stream to the client into a hash table. This table is shared by all server handlers enabling broadcasting and multicasting of messages from one user to any set of others.

On the client side, all networking is handled through a specially designed client thread that listens for messages and sends data to the servers. Thus, the client objects serves as a sort of proxy server that abstracts the network into a Java object.

Dynamic Compilation, Binding, and Linking

Modules in Fuse-N are individual object files with several associated helpers. The main element of a module is the ModuleBean, which includes all code specific to the functionality of that module when represented as a block within a data flow. The ModuleBean handles input to output mappings and then passes its data to its associated connector objects. Connector objects are essentially routers that take inputs from zero or more sources and send them to zero or more targets. An input connector will take data from connectors linked to it and pass it into its associated ModuleBean. Likewise, an

output connector will take data from its associated ModuleBean and pass it to the connectors to which it is linked. When a student compiles a piece of code in Implement mode, the source code is sent to the Java servlet, which starts a native compilation thread. The client then dynamically loads in the new class file and binds it to the visualization in the learning tool and to the block in the data flow. In order to maintain references in the face of dynamically changing instances, the ModuleBean object is encased in a static ModuleBeanWrapper object which maintains the data that should be saved across compilations, such as input queues and connector references. When new connections are created in a data flow, the input and output connectors involved add the appropriate connectors to their inputs and outputs.

Bibliography

[Ber97] Berners-Lee, Tim. "WWW Addressing Overview." *http://www.w3.org/pub/WWW/Addressing/*. World Wide Web Consortium, 1997.

[Boy97] Boyd, Nathan D. T. "A Platform for Distributed Learning and Teaching of Algorithmic Concepts." MIT Thesis. 1997.

[Bro+84] Brown, Marc H. and Robert Sedgewick. "A System for Algorithm Animation." SIGGRAPH Proceedings: Volume 18, Number 3. July 1984. Pages 177 – 186.

[Bro91] Brown, Marc H.. "Zeus: A System for Algorithm Animation and Multi-View Editing." In IEEE Workshop on Visual Languages, pages 4-9, October 1991. Also appeared as SRC Research Report 75. (Postscript).

[BGG98] Brown Computer Graphics Group. "Exploratory." *http://www.cs.brown.edu/research/graphics/research/exploratory/exploratory.html* 1998.

[Coh94] Cohen, Abbe. "Inessential Zephyr." *http://www.mit.edu:8001/afs/sipb/project/doc/izephyr/html/izephyr.html.* The Student Information Processing Board, 1994.

[Dol97] Dollins, Steven C. "Interactive Illustrations." *http://www.cs.brown.edu/research/graphics/research/illus/*. Brown University Department of Computer Science, 1997.

[Gol+96] Goldberg, Murray W., et. al. "WebCT – World Wide Web Course Tools." *http://homebrew.cs.ubc.ca/webct/*. University of British Columbia Department of Computer Science, 1996.

[Gos+96] Gosling, James and Henry McGilton. "The Java Language Environment: A White Paper." Sun Microsystems. May 1996.

[Hae88] Haeberli, Paul E. "ConMan: A Visual Programming Language for Interactive Graphics." SIGGRAPH Proceedings: July 1988. Pages 103 – 111.

[Hic97] Hickley, Tim. "Jscheme: an Applet for Teaching Programming Concepts to Non-Majors." *http://www.cs.brandeis.edu/~tim/Packages/Jscheme/Papers/jscheme.html* Brandeis University, Michtom School of Computer Science, 1997.

[IBM98] International Business Machines, Inc. IBM Data Explorer (DX). *http://www.almaden.ibm.com/dx/* 1998.

[Lau93] Laurillard, Diane. Rethinking University Teaching: a framework for the effective use of educational technology. Routledge: London. 1993.

[Law+92] Lawrence, Andrea W., John T Stasko, and Eileen Kraemer. "Empirically Evaluating the Use of Animations to Teach algorithms" *ftp://ftp.cc.gatech.edu/pub/gvu/tech-reports/94-07.ps.Z* Technical Report GIT-GVU-94-07. Georgia Institute of Technology College of Computer Science. 1994.

[Mah97] Mahler, Thomas. "webLISP" *http://www.vordenker.de/weblisp/main.htm*

[Por98] Porter, Brandon W. "Educational Fusion: An Instructional, Web-Based Software Development Platform." MIT Thesis. 1998.

[Rob96] Roberts, Eric. "Strategic Directions in Computer Science Education". ACM Computing Surveys 28(4), December 1996.

[SM97] Sun Microsystems, Inc. "Java Servlets: The Power Behind the Server" *http://java.sun.com/products/java-server/servlets/index.html.* Sun Microsystems, Inc., 1997.

[SM98] Sun Microsystems, Inc. "JDK 1.1.6 Documentation" *http://java.sun.com/products/jdk/1.1/docs/index.html.* Sun Microsystems, Inc., 1998.

[Sta96] Stasko, John T., " Using Student-Built Algorithm Animations as Learning Aids", Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-GVU-96-19, August 1996.

[Sta+92] Stasko, John T and Eileen Kraemer. "A Methodology for Building Application-Specific Visualizations of Parallel Programs." *ftp://ftp.cc.gatech.edu/pub/gvu/tech-reports/92-10.ps.Z* Technical Report GIT-GVU-92-10. Georgia Institute of Technology College of Computer Science. 1992.

[Su78] Sussman, Gerald Jay and Guy Lewis Steele Jr. "The Revised Report on Scheme, a dialect of Lisp." MIT Artificial Intelligence Memo 452, January 1978.

[Tel94] Seth Teller. "NSF Career Development Plan." *http://graphics.lcs.mit.edu/~seth/proposals/cdp.ps.* MIT Department of Electrical Engineering and Computer Science, 1994.

[Try97] Samuel Trychin. *Interactive Illustration Design.* Brown University Computer Graphics Lab, 1997.

[Ups+89] Upson, Craig, Thomas A. Faulhaber, Jr., David Kamins, David Laidlaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam. "The Application Visualization System: a Computational Environment for Scientific Visualization". IEEE Computer Graphics and Applications, Vol. 9, Num. 4. p.30-42. July 1989.

[Zac+97] Joe Zachary, et. al. "Hamlet Project." *http://www.cs.utah.edu/~hamlet/.* University of Utah Department of Computer Science, 1997.