Flatland: Rapid Prototyping of Distributed Internet Applications

Harry Chesley, Greg Kimberly, Steve White, Anoop Gupta, Steve Drucker

8/9/2001

Technical Report
MSR-TR-2001-73

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Flatland: Rapid Prototyping of Distributed Internet Applications

Harry Chesley, Greg Kimberly, Steve White,

Anoop Gupta, Steve Drucker
Microsoft Research
Redmond, WA. 98052 USA

ABSTRACT

Computer intra- and internets are widely used for client-server application such as web browsers. With the exception of e-mail, however, the same networks are seldom used for distributed, client-client or client-server-client applications. Such applications are difficult to develop and debug, and require a supporting infrastructure that is not readily available from existing systems. Flatland is a rapid prototyping environment that provides the underlying infrastructure and makes it easy to create and debug distributed internet application prototypes. In addition to the infrastructure needed for a distributed application, Flatland includes safe implementations of the most common sources of distributed application bugs – asynchronous operation and updating. Flatland also supports streaming audio-video and down-level clients.

Keywords

Internet, distributed applications, rapid prototyping, streaming video, distance learning.

1 INTRODUCTION

Contemporary computers and networks are increasingly able to support distributed, real-time applications. Yet, few of these applications have been delivered to date. Part of the reason for this is the difficulty of developing asynchronous, multiple machine software. Development requires additional infrastructure, and often leads to difficult to bugs that are difficult to reproduce and fix due to their asynchronous nature.

Since there is relatively little pre-existing experience with distributed applications to extrapolate from, it is also an area that benefits greatly from rapid prototyping. Yet, again, the infrastructure is not there for rapid prototyping, and asynchronous issues can overwhelm the developer, diverting resources from prototyping the application at hand.

Flatland brings together a number of existing technologies with an architecture than reduces the chances of asynchronous issues arising. It leaves the distributed application developer to the issues specific to their application, without needing to spend time on generic distributed software concerns.

V-Worlds, a platform developed by the Virtual Worlds Group in Microsoft Research, provides the core distributed persistent object facilities of Flatland. DHTML as implemented in Internet Explorer provides the user interface aspects. JScript provides the implementation language. And NetShow provides streaming audio-video.

Flatland has been used to develop several distributed applications. These include a text/icon chat system, a distance learning application, and a research tool for investigating social dilemmas.

2 THE PROBLEM

Over the past twenty years, the use of rapid prototyping to explore an application design space has gained increasing acceptance as a means of software design. Meanwhile, the move from single-computer applications to distributed applications has opened the door to vast new potential uses for computer systems. But at the same time, it has increased the complexity of those applications and the difficulty of prototyping and developing them.

2.1 Rapid Prototyping

Traditionally, most computer applications have been local, single-machine applications. Rapid prototyping has proved to be a valuable technique for evolving software design. [9] Over time, software development facilities and languages (such as Microsoft Visual Basic and Smalltalk) have developed to allow rapid experimentation with new software concepts and designs. These rapid prototyping environments allow product planners, software developers, and user interface researchers to develop and test multiple hypothesizes before committing to a single final design.

Recent work in user-centered design is particularly well suited to developing applications and user interfaces via rapid prototyping. Using this methodology, feedback from actual users is central to the design process, and the ability to easily and rapidly incorporate that feedback into the software is critical.

Distributed applications require additional infrastructure over single-user, single-machine applications, making rapid prototyping more challenging.

Flatland brings together a number of underlying technologies to enable rapid prototyping, and adds an architecture that makes distributed applications in particular easy to build.

2.2 Distributed Applications

With the advent of the Internet, there are many opportunities for distributed applications in several areas:

- Distance learning
- Computer-supported cooperative work (CSCW)
- On-line communities
- Multi-player games

Distributed applications, however, are more difficult to prototype and debug. In addition to the more complex logistics of distributed applications – with at least twice as many components as a stand-alone application – they are intellectually more challenging to design and debug. The human mind simply works better on single-threaded problems.

2.2.1 *Infrastructure*

The first barrier to distributed application development is infrastructure. Well-established protocols exist for client-server applications, but not for distributed, client-server-client applications. While it is possible to create distributed applications using existing protocols like IRC or DCOM, it is not an easy task.

Flatland provides the infrastructure so that application developers can concentrate on the specific problem at hand, not on developing the supporting technologies. It does this by building on top of the existing V-Worlds platform.

2.2.2 *Asynchronous Operation/Mutual Exclusion*

The most common source of bugs in distributed applications has to do with asynchronous operation. Without proper facilities for mutual exclusion, and without their careful application, programs using multiple simultaneous threads of execution often develop cross-thread interactions that are extremely difficult to understand and reproduce.

Another aspect of asynchronous operation is update propagation. When a change is made in one part of the system, that change must propagate to the rest of the distributed application, causing appropriate user interface changes for all users. This is also known as the update model.

Flatland provides a pair of update models that centralize the process of updating objects and propagating the changes. These update models are carefully designed to limit the problems that arise from asynchronous operation.

2.2.3 *Audio-video Stream Synchronization*

When streaming audio-video is used in an application, an additional complication arises: due to network latencies, different clients may receive the audio-video stream at different points in time, even though their connections with other network elements are essentially real-time. This requires synchronization between the audio-video stream and other aspects of the application.

Flatland includes an update model that automatically defers updates until the appropriate point in time as defined by the received audio-video stream.

3 COMPONENTS

The Flatland system is build on top of a number of existing technologies. V-Worlds provides real-time client-server-client communication. DHTML provides easy construction of user interface elements. Active scripting provides scripting languages for rapid prototyping. NetShow provides audio-video streaming. All of these are tied together by the Flatland architecture.

3.1 V-Worlds

V-Worlds [5] is a platform for distributed client-server-client applications. It provides automatic transfer of object properties, as well as remote procedure calls, among a set of communicating clients and a single server. It provides for persistent storage of the objects within the system. It also incorporates a mechanism to limit the communications overhead by restricting which clients are notified of changing in which objects.

3.1.1 *Objects*

V-Worlds is a persistent object system. Data and behavior is encapsulated within objects. The objects are automatically moved from client to server to client as needed, and are archived on the server for persistence across sessions.

Objects include properties that can be simple data types (number, string, etc.), compound data (list, map, etc.), or other objects. Properties can be marked on a case-by-case basis whether they should be distributed or only maintained locally.

Behavior is provided in the form of methods on the objects. Methods can be marked to execute on the server, on the client, or on either. If a client-side method is invoked from the server, it is executed in parallel on all of the connected clients.

V-Worlds provides dynamic inheritance. Rather than taking their methods and initial property values from a class definition, V-Worlds objects take them from an exemplar object. This provides more flexibility for dynamic updates than a class/instance model.

3.1.2 *Avatars*

The user/client is represented within V-Worlds using an Avatar object. This object includes login information, current connectivity information, and anything else that is specific to a user. It also helps to determine what information needs to be transferred to the associated client.

3.1.3 *Rooms*

In order to limit the amount of client-server-client communication, all objects in V-Worlds must exist within the contents list of a Room object. Server objects are only distributed to clients if the client's Avatar object is in the same room as the object. This allows multiple sets of Avatars to maintain shared accessible object pools, without incurring the overhead of every client hearing about every object in the system.

3.1.4 *Summary*

V-Worlds provides Flatland with a distributed, persistent object system that hides the details of client-server-client communication and object archiving. This is the core of any distributed application.

3.2 DHTML

Dynamic HTML (DHTML) is composed of the Internet standards HyperText Markup Language (HTML) 4.0, Cascading Style Sheets (CSS) 1.0, and the associated document object model. Taken together, these provide a powerful system for quickly building and modifying an application user interface. This is the core of Flatland rapid prototyping of user interface designs.

In addition to the current facilities implemented in the current version of Internet Explorer and used in existing Flatland work, these standards and their implementations continue to evolve and expand, constantly making more user interface capabilities available for prototyping.

3.3 Scripting

Several scripting languages are now available for scripting both HTML pages and V-Worlds object methods. JScript, ECMAScript, VBScript, and Perl are all available.

Scripts are used in Flatland for two purposes: Within the V-Worlds objects, scripts implement the semantics of an application element. Within DHTML, they implement the user interface aspects.

For Flatland, we have used JScript (a superset of ECMAScript) almost exclusively, with small amounts of VBScript where required to capture error exceptions.

3.4 NetShow

Microsoft NetShow provides the streaming video facilities required by some Flatland applications. NetShow encoding and playback can be controlled from script from within Flatland.

NetShow also provides the ability to encode data within the audio-video stream. Flatland makes use of this capability for two purposes. First, it includes time synchronization information in the stream so that other Flatland events can be synchronized to the audio-video stream. Second, it includes information to support down-level browsers that do not have access to the V-Worlds part of the system.

3.5 Limitations

Building Flatland on top of a large and complex set of technologies is like standing on the shoulders of giants. You can go and see further, but you're limited by where the giants are willing to take you.

Flatland was built primarily on V-Worlds 1.0 and Internet Explorer 4.01 running on the Windows operating system (95, 98, or NT). While it is theoretically possible to extend these systems to other platforms, such a task is well beyond the work described here.

However, in order to make the remote learning application of Flatland more widely useable, we did build a "down-level client," that allows other platforms to watch, though not participate, in the system. See below for more details.

4 FLATLAND ARCHITECTURE

The Flatland architecture pulls all these components and goals together into a single coherent design that minimizes the amount of effort required to construct and modify distributed applications, while maximizing the range of those potential applications. Flatland implements the foundation of a distributed application, including most of the components that traditionally cause obscure and difficult to locate bugs. But it provides an open-ended script-based environment for semantic development, and a powerful, DHTML-based environment for UI development.

4.1 Distributed Model/View

The Flatland architecture divides the application into discrete elements. Each element is further divided into a distributed object that exists within V-Worlds, and a user interface object that exists within DHTML in an Internet Explorer window. This is a natural evolution of the Smalltalk model/view/controller paradigm [6] in a distributed environment, with the view and controller combined in the DTHML object.

The model object within V-Worlds can be thought of as existing across the server and all participating clients. The model object is responsible for maintaining the current semantic state – as opposed to the user interface representation of that state – for distributing changes among the server and the clients, and for persisting the state across multiple sessions.

The DHTML user interface object is implemented using scriptlets. A scriptlet is an Internet Explorer feature, similar to a frame, that encapsulates a user interface implementation as a reusable element. This allows implementation of the UI for one element of the application independent of the other elements, as well as reuse of elements between different applications.

The client portion of the model object and the user interface scriptlet maintain references to each other to facilitate communication. Both objects implement COM OLE automation, allowing script level access to objects, properties, and methods.

4.2 Update Model

Flatland standardizes the update model for application elements in order to avoid synchronization and update issues described earlier. Two models are available, for data-driven immediate updates and for event-driven updates that are synchronized with the associated audio-video stream.

4.2.1 Data-driven Update Model

The simplest update model in Flatland is used for elements that do not require synchronization with the audio-video stream, or for elements that will be used in applications without such a stream. In this approach, properties of the model object are changed on the server. V-Worlds then automatically distributes the changes to the clients. Then an update method is called on the user interface object to display the changes to the user. Updating the model object on the server, plus a single semaphore on the model object, preclude changing or accessing the model in an invalid state.

In more detail, the process takes place as follows:

1. Something triggers a change in an application element. Most often, this is an action by one of the users, interpreted by the user interface object of the element.

2. A server-side method is called to update the model object. Server-side methods are single-threaded in V-Worlds, ensuring mutual exclusion of updates.
3. The server-side method calls BeginUpdate(), which sets the exclusion semaphore on the model object.
4. The server-side method makes a set of mutually consistent changes to the properties of the model object.
5. The method calls EndUpdate(), which clears the semaphore, and initiates the user interface update process.
6. An update method is called on the user interface objects associated with this application element on all of the clients. This method is responsible for displaying the new state of the element to the users.

The semaphore is used to ensure that access is not made to the model object when it's in an inconsistent state. This will not happen on the server, since the server is single-threaded, but client-side methods on the model object or on the user interface object could attempt to access the model object in the middle of an update. These methods need to check the semaphore before access the model object. In most cases, however, the user interface update method is the primary point of reference into the object, and the Flatland architecture will not call this method if the object is not consistent.

This update model is simple to use, and ensures proper mutual exclusion of model object changes and appropriate propagation of user interface updates. To use it, developers only need to add a server-side model object property update method for the V-Worlds object that implements the semantics of the element and a user interface update method for the scriptlet that updates the interface.

4.2.2 Event-driven, Synchronized Update Model

When updates need to be synchronized with the audio-video stream, a different update model is available. In this case, changes are packaged into events that are deferred until the appropriate point in time, and then an event-specific update method is invoked in the user interface object.

In order to provide synchronization, time-stamps are periodically inserted into the audio-video stream. Every client receiving the stream can then maintain its temporal position relative to the current stream. This information is used to set the time of newly created events in the originator, and to determine how long to defer events in the receiver.

The synchronized event process takes place as follows:

1. Something triggers a change in an application element. Most often, this is an action by one of the users, interpreted by the user interface object of the element.
2. An event object is created which includes the event type, the time relative to the audio-video stream, and any parameters that further define the event.
3. The event is sent to the server and redistributed to all of the clients.
4. Each client queues the event until the appropriate time arrives for it to be invoked. This might be immediate if the time has already passed, or might be up to several seconds in the future.
5. When the time arrives, a event-type-specific method is invoked on the user interface object that is passed the event parameters. This method updates the user interface as appropriate.

This update model is more complex to implement than the simple model described in the previous section, but it allows for synchronization with the audio-video stream. The developer must define one or more application element-specific events, and create the associated update methods.

4.3 Composite Elements

Most Flatland application elements are simple, stand-alone items. However, it is also possible to create composite elements with nested sub-elements. The most common use of nest elements is to create window layouts.

When a user is connected to Flatland, they see a window that corresponds to the Room where their Avatar is located. This window contains one top-level application element, also known as a window layout. That element generally contains a number of sub-elements.

Another use of composite elements is sharing screen space between more than one simple element. In the remote learning application of Flatland, for example, a slide area shows a sequence of slides to the user. Each slide is itself an instance of an application element, and the slide area is a composite element.

4.4 Authorization

Another common element of distributed applications is the need for authorization facilities. There are usually differences between what the general public and the owners of the application are allowed to do. Flatland provides four levels of authorization: audience, presenter, authorizer, and room creator.

(Note: The terms used originated in the remote learning application of Flatland, hence the bias toward “presenters” and “audiences.”)

4.4.1 Audience Authorization

The most basic level of authorization is audience – the ability to be in the room at all. Without this authorization, a user cannot enter a room. Audience authorization is on a room-by-room basis.

4.4.2 Presenter Authorization

A presenter is given more control over the application elements, together with the ability to create and edit them.

The details of what a presenter can do beyond what an audience member can do are specific to the individual application element. Presenter authority is on a room-by-room basis.

For example, in the distance learning application, new slides can only be created by a presenter, and only a presenter can change the video source.

4.4.3 Authorizer Authorization

Authorizers can change the authorization of themselves and other users within a given room. They can enable a specific user to be audience members, presenters, and/or authorizers. They can also enable “anyone” to be one of these. Giving “anyone” authorization is most commonly used to open the doors to the room and let everyone in once the room is ready for visitors.

4.4.4 Room Creation Authorization

The final level of authorization is the ability to create new rooms. When a new Flatland site is created, it starts with no rooms, and only the site administrator is authorized to create new rooms. The administrator can then create new rooms and/or authorize other people to do so.

4.5 Down-level Clients

Full Flatland functionality is available only under Internet Explorer 4.01 running on the Windows operating system. However, many Flatland applications, especially those in distance learning, need the ability to operate with other browsers and operating systems, even if it is in a non-interactive fashion.

Flatland provides this ability for applications that use an audio-video stream. Down-level clients are available that extract events from the audio-video stream and update the local HTML page as appropriate. No interaction with the distributed, V-Worlds based system is available, but at least users with other systems can watch the activity.

The same mechanism is employed for archival playback of a presentation.

5 APPLICATIONS

Flatland has been used for a number of specific applications. These include text and graphic chat systems, distance learning, and social dilemma research.

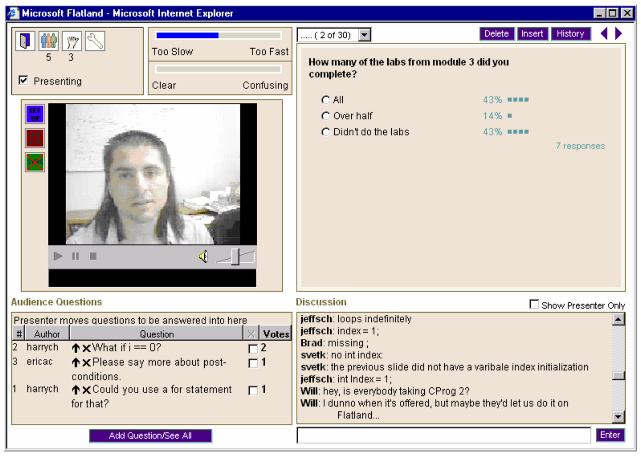
5.1 Heads Chat

Perhaps the simplest Flatland application is a text chat facility that is complimented by head icons that represent the users. The heads allows the users more expression than pure chat. They can move to form groups, look at different parts of the screen, and the presenter can create sub-areas to divide the audience into discussion or voting groups.

Heads chat is an example Flatland application that gives Flatland developers a jumping off point to create their own applications.

5.2 Remote Learning

The remote learning application of Flatland combines NetShow streaming audio and video with a collection of audience feedback mechanisms that allow the presenter to receive both solicited and unsolicited responses from the viewers. Figure 1 shows the main screen layout, as seen by a presenter.



How many of the labs from module 3 did you complete?

All	43% ■■■■
Over half	14% ■
Didn't do the labs	43% ■■■■

7 responses

Presenter moves questions to be answered into here

#	Author	Question	Votes
2	harrych	What if i == 0?	2
3	ericac	Please say more about post-conditions.	1
1	harrych	Could you use a for statement for that?	1

jeffsch: loops indefinitely
jeffsch: index = 1;
Brad: missing ;
svetk: no int index:
svetk: the previous slide did not have a varibale index initialization
jeffsch: int Index = 1;
Will: hey, is everybody taking CProg 2?
Will: I dunno when it's offered, but maybe they'd let us do it on Flatland...

Figure 1 – Flatland Presenter Layout

Figure 2 shows the Flatland components and their relationships. A presenter communicates with a number of audience members using NetShow video and Flatland. The audience, in turn, can pass questions, answers, and requests back to the presenter via Flatland.

5.2.1 User Interface

Figure 1 shows the main Flatland window layout, as seen by the presenter. The audience sees a similar view, but without many of the controls and buttons.

The middle left section of the layout contains the video of the presenter, provided using Microsoft NetShow 3.0 [7]. Any Flatland participant with a video feed could present, but in these studies only the instructor did.

The upper right section of the window contains slides and questions, as defined by the presenter. This area can include slides generated by Microsoft PowerPoint, simple text slides, and audience Q&A slides that allow the audience to vote by selecting one of the answers to a multiple choice question. The presenter can also use a ”pointer” to indicate specific sections of the slide during the presentation.

The presenter controls the selection of the currently displayed slide. A History button above the slide area, however, generates a separate window with the entire set of slides for the current presentation. This allows any viewer to browse the slides not currently being displayed in the main window.

Presenter controls in the slide area include facilities to select the slide to be displayed, using either the ”next” and ”previous” arrow buttons on the top right or the table-of-contents pop-up on the top left. There are buttons to edit or delete the current slide. A presenter can also create a new slide on the fly and insert it in the presentation.

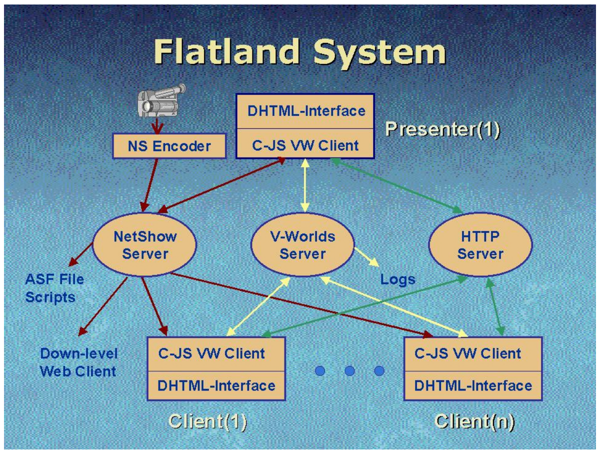


Figure 2 – Flatland System

Below the slides, on the right, is a text chat area. This allows free-form communication between audience members or between the audience and the presenter. Interactive chat gives audience members a strong feeling of the presence of other participants, and can be invaluable for resolving last minute technical problems that audience members may encounter. This window also reports when people join or leave a session.

Although free-form chat is valuable in providing an open and unrestricted communications channel, it can easily become overwhelming. For questions specifically directed at the presenter, a separate question queue is provided to the bottom left of the window. In this area (hereafter called the Q&A window), audience members can pose questions for the presenter. They can also add their support to questions posed by others by incrementing a counter. This voting capability could reduce duplicate questions and help a presenter decide which question to address next.

Finally, the upper left area of the window provides several lighter weight feedback mechanisms. On the right are two checkboxes that allow the audience to give continuous feedback on the speed and clarity of the presentation, displayed as a meter on the presenter window. On the left are buttons to leave the presentation, to show a list of audience members, and to raise a hand for impromptu audience polling. A pop-up, floating ”tool tip” shows a list of members with their hands raised if the cursor is left over the hand icon. The same information is also displayed in the pop-up audience member list.

5.3 Social Dilemma

To be added.

6 CONCLUSIONS

The widespread availability of the Internet has led to many opportunities for distributed Internet applications. However, developing distributed applications is challenging, and has few existing examples to build upon. This adds greatly to both the desire for, and difficulty of, rapid prototyping in this environment.

Flatland builds upon the existing facilities of V-Worlds, DHTML, active scripting, and NetShow to provide an architecture that facilitates rapid prototyping by supplying pre-built infrastructure and elements that perform the basic and most error-prone elements of any distributed application, while allowing considerable freedom of implementation.

Flatland has been used to develop several applications, including a remote learning system and a social dilemma workshop. Microsoft Research is continuing to develop new uses for it.

7 REFERENCES

1. Isaacs, E.A., Morris, T., and Rodriquez, T.K., A Forum For Supporting Interactive Presentations to Distributed Audiences, *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '94),* October, 1994, Chapel Hill, NC, pp. 405-416
2. Isaacs, E.A., Morris, T., Rodriquez, T.K., and Tang, J.C.(1995), A Comparison of Face-To-Face and Distributed Presentations, *Proceedings of the Conference on Computer-Human Interaction (CHI '95),* Denver, CO, ACM: New York, pp. 354-361.
3. Isaacs, E.A., and Tang, J.C, Studying Video-Based Collaboration in Context: From Small Workgroups to Large Organizations, in *Video-Mediated Communication*, K.E. Finn (ed.), 1997, A.J. Sellen & S.B. Wilbur, Erlbaum: **city, state** , pp. 173-197
4. Begeman, M., Cook, P., Ellis, C., Graf, M., Rein, G. & Smith, T., 1986. Project Nick: Meetings augmentation and analysis. Proceedings of CSCW'96. Revised version in *Transactions on Office Information Systems, 5,* 2, 1987.
5. Vellon, M., K. Marple, D. Mitchell, and S. Drucker. The Architecture of a Distributed Virtual Worlds System. Proceedings of the 4th Conference on Object-Oriented Technologies and Systems (COOTS). April, 1998.
6. Glenn E. Krasner and Steven T. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80", in Journal of Object-Oriented Programming, 1(3), pp. 26-49, August/September 1988.
7. Microsoft NetShow. Available at http://www.microsoft.com.
8. Microsoft INet SDK. Available at http://www.microsoft.com.
9. Tanik, M.M., Yeh, R.T. Rapid Prototyping in Software Development. Computer, May, 1989.