Dynamicity Issues in Broadband Network Computing

Steven Gutfreund
Jose Diaz-Gonzalez
Russell Sasnett
Vincent Phuah

Distributed Multi-Media Applications Project
GTE Laboratories, Inc.
40 Sylvan Road
Waltham, MA 02254 USA
jdiaz@gte.com, sgutfreund@gte.com
vphuah@gte.com, rsasnett@gte.com

1.0 Abstract

A compound multi-media document consists of a mix of continuous media elements (audio, video, instrument sensors, etc.) and computational elements that display, chart, record, and process the media elements. In a network environment, where these elements exist at different locations on the communication fabric, there are obvious synchronization problems. However, there is also a class of problems we call dynamicity issues, which are concerned with the need to: reconfigure the network, add and remove connections that bind the elements, deal with mobile elements. In this paper we characterize the different types of dynamicity that occur in networked compound documents and present a top-level architecture for managing dynamicity.

2.0 Background

For the last two years, our group has been involved in the design of multi-media environments. This work has consisted of two parts:

1. We have been an active participant and contributor to the design and development of the Athena Muse multi-media authoring system by the Visual Computing Group at MIT Project Athena [Hodge89]. Our collaboration also consisted of the loan of one of our group members, one of the original designers of Muse, to the Muse project.

2. The construction, in cooperation with other research groups at GTE Laboratories, of a hybrid, digital broadband network capable of supporting multi-media connectivity among our offices. This hybrid network utilizes a (GTE proprietary) SONET-compatible digital switch that was used to provide video connectivity to/from multiple sources, a computer controllable ISDN switch for voice connections, and an Ethernet network for data connections. We have ported and made modifications to Muse to make use of this network and created prototypes with Muse of cooperative documents for multi-user, multi-media applications.

3.0 The Next Effort

Our next effort involves the migration from a three network interconnect (ethernet, SONET, ISDN) to a single unified ATM-based [Miner89] gigabit LAN. In this configuration we will be able to provide a much tighter degree of integration in our compound multi-media applications. For example, we will be able to provide real-time synchronization between video clips and computer simulations, or to have a computer visualization driven by real-time instrument sensors connected to a laboratory experiment.

In order to create such applications one may have to combine many different continuous (e.g. audio, video), discrete (e.g. data bases), and computational (e.g. visualization programs such as AVS [AVS91]) elements. Each of these elements may reside on a different node. The bandwidth, latency, and quality-of-service (e.g. error rate) between nodes can have a significant impact on the design and performance of an application. Furthermore, various *dynamic* changes in the network, software topology, loss/addition of video elements, and movement of the user dictate the need for a flexible element interconnection architecture. However, before creating an interconnection architecture for multi-media applications we need to examine and understand the different aspects of these dynamic network changes.

4.0 Dynamicity

Dynamicity in distributed multi-media applications exist at four levels:

1. Transportability
2. Mobility
3. Reconfigurability
4. Plasticity

4.1 Transportability

Transportability involves the need to move applications from one network to another. When an application moves one has to re-bind it to the various media, data, and compute elements that participate in the application. This re-binding can be more complicated than what can be accomplished with a simple directory service. Because of the different topology, throughput, and line quality at the new location, different choices may have to be made concerning which compute and media elements will be part of the application.

4.2 Mobility

Mobility is an issue concerned with the migration of active applications. One would like to be able to download a running application from an office workstation to a multi-media notebook connected via a cellular phone, and then move to a new location (e.g. home) where the application would be transferred to the home system. If the application is shutdown then this is a transportability issue. However, if the application needs to be active during the migration phase, then we have a mobility issue. Mobility issues are more difficult than transportability issues in that the binding of the multi-media document needs to be constantly re-checked. In a transportable document, binding only needs to be performed at start-up.

4.3 Reconfigurability

The location in the network where the user is running the system is only one element of dynamics. Another involves the dynamic changes to the topology of the network and the addition and loss of compute and media resources. A reconfigurable multi-media application can dynamically re-organize its granularity of distribution to make use of more compute resources. It is also capable of discovering better links to equivalent classes of media and data resources.

4.4 Plasticity

Compound multi-media applications are usually not created from scratch. They involve the assembling of various pre-built elements such as video, audio, visualization programs and databases. Pre-built elements come with their own set of user interface widgets. When creating the user interface for the compound application, one does not want to merely concatenate the underlying user interfaces. Instead one would like to create one unified interface. Such an interface could synchronize related controls on several different underlying components. Additionally, a user should be able to customize an interface to make use of more sophisticated interaction devices that are available at different user stations on the network.[1] We call such customizable interfaces plastic interfaces.

5.0 SHOWME

SHOWME (SHared Object-oriented Workbench for Media Elements) is a multi-media environment that attempts to address the dynamicity issues that arise in distributed applications.

SHOWME applications consist of a set of *elements* that are distributed over a broadband network. An element is a single uniform homogenous term to describe the items that are composited to form a SHOWME application. Elements can be continuous data resources such as video or audio sources. Alternatively, an element can be a discrete data source such as a data base. Elements can act as sources or sinks of information in SHOWME composite documents. For example, a video element can be the source of an application and the sink can be a database (figure 1).

Elements form the fundamental unit of distribution of SHOWME applications. In other words, a SHOWME application is constructed by assembling a set of elements, where the elements reside at different locations in a distributed environment. Elements provide a single uniform component for describing and specifying an application.

Elements can be computational. That is, while many elements in a SHOWME application are mere data sources or sinks, some are computational elements that transform data. Data is piped through a computational element from its source to its sink. These sources or sinks can in turn be computational elements. Thus, one can create applications which con-

1. With the acceptance of the X window environment, it has become much easier to detach an application from its user interface and attach a new set of virtual widgets.

sist of a stream from a video source, computational transformational elements that extract events from the video stream, and windows that act as sinks to display the video and the extracted event stream.

Figure 2 shows the basic SHOWME architecture.The binding of elements is handled by the Resource Dispatcher. The Dispatcher is a hybrid of Linda [Carri89], the x-kernel [Peter90], and the Message Backplane [Reiss90]. It provides a distributed shared memory called tuple space. Objects in tuple space can be located by pattern matching, range queries, or user procedures. Elements of a multi-media application register themselves in the shared memory. Dynamic binding of elements is performed by pattern matching and queries to the Dispatcher (figure 3). The actual data flow between continuous media and compute elements can either be through the shared memory or over out-of-band channels (e.g., ATM virtual circuits [Miner89]) depending on the bandwidth needs of the connection.

Connected to the Dispatcher are the Dispatch Analyzer and Dispatch Controller. The Analyzer is used to conduct performance tests for different Resource Dispatcher implementations. It collects throughput, latency, and error rate statistics and can chart performance for different packet and burst sizes. The Dispatch Controller is used for tuple space maintenance. It can browse, add, and remove tuple spaces in an individual Dispatcher. It can also be used to change hashing, layout, and storage parameters associated with a tuple space. In the future, it will be used to manage data segmentation among multiple Dispatchers.

Since elements are frequently pre-built, and sometimes are commercial off-the-shelf software packages, a wrapper must be applied to elements to re-direct their data sources and sinks to tuple space. This is performed by the Bonds. There are four different types of information that a Bond will forward to an element from tuple space: control, data, widget, and script.

Control data are tuples that are used for starting and stopping elements. A tuple can be placed in tuple space to start the execution of an element. Other control tuples in tuple space will specify where to obtain the executable of the element, which node will run it, what the command parameters are, and what the X-resource defaults are.

Data tuples are used to specify the input/output files that the element uses (e.g. standard input/output). The data tuple can either directly contain the entire contents of the file, or specify a handle for an out-of-band ATM circuit (typically a direct channel to a persistent data base).

Widget tuples provide values for the widgets that the element needs. Each action that the user performs on his interface (e.g. a button press) will result in a widget tuple being placed in the Dispatcher. The Bond-wrapper looks for these tuples and forwards them to the element disguised as a normal X-event.

Lastly, the user can write scripts in a high level language (a derivative of the Muse script language [Hodge89]). Scripts are typically used to automate widget actions. For example, a script can produce a stream of widget tuples that simulate user actions. Scripts can also be used to produce control and data tuples, however there are limits to the scope of the scripting language.

6.0 Conclusions

Pattern-directed binding is very flexible. Binding between elements occurs at run-time allowing applications to be both mobile and transportable. Furthermore, compute tasks can be registered in shared memory and dynamically dispatched to compute servers - allowing a degree of reconfigurabily of an application based on the presence of more compute resources. Media can be bound by type, i.e. if one is on a network link that supports non-compressed HDTV video one can bind directly to an HDTV feed. Otherwise, the pattern-based query may result in a binding to a lower grade of service such as compressed NTSC. Plastic interfaces can be created by either substituting widgets that produce equivalent widget tuples or by using scripts to simulate the invocation of widget controls.

We believe that the SHOWME architecture shows great promise in being able to meet the dynamicity goals outlined in the previous sections. We have currently implemented the central parts of Dispatcher system and preliminary performance studies have been very promising.

7.0 Acknowledgments

We wish to thank Dean Casey and Harry Mussman for their contributions in setting up our SONET video network. We would also like to thank Dave Decker, Bill Griffin, and Jerry Jones for providing the management support and climate conducive to this research.

References

[AVS91] AVS, Scientific visualization package, Stardent Computer Corp, Newton, MA.

[Carri89] Carriero, N., and Gelernter, D., "How to Write Parallel Programs: A Guide to the Perplexed," *ACM Computing Surveys*, Vol. 21, No. 3, pp. 323-357. September 1989.

[Gutfr91] Gutfreund, S., "Integrating Distributed Visualization Systems with Multi-Media," *Workshop on Computer Graphics in the Network Environment, Proceedings of SIGGRAPH '91*. July 1991.

[Hodge89] Hodges, M., Sasnett, R.M., and Ackerman, M.S., "A Construction Set for Multimedia Applications," *IEEE Software*, Vol. 6, No. 1, pp. 37-43. January 1989.

[Miner89] Miner, S., "Broadband ISDN and Asynchronous Transfer Mode (ATM)," *IEEE Communications Magazine*, Vol. 27, No. 9, pp. 17-24. September 1989.

[Peter90] Peterson, L., Hutchinson, N., O'Malley, S., and Rao, H., "The x-kernel: A Platform for Accessing Internet Resources," *IEEE Computer*, Vol. 23, No. 5, pp. 23-33. May 1990.

[Reiss90] Reiss, S., "Integration Mechanisms in the FIELD Environment," *IEEE Software*, Vol. 7, No. 4, pp. 57-66. July 1990.

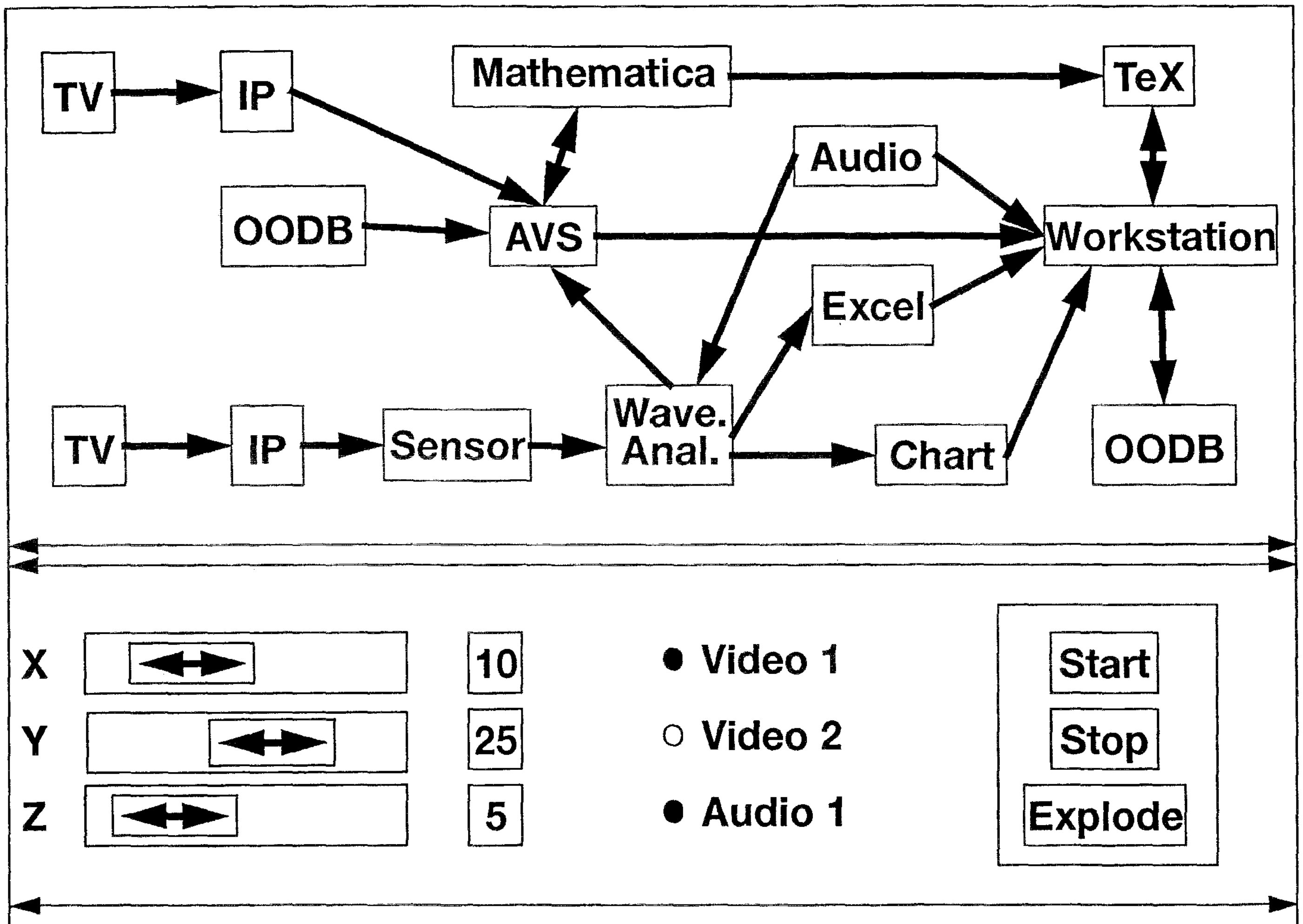


Figure 1: A Distributed Compound Document

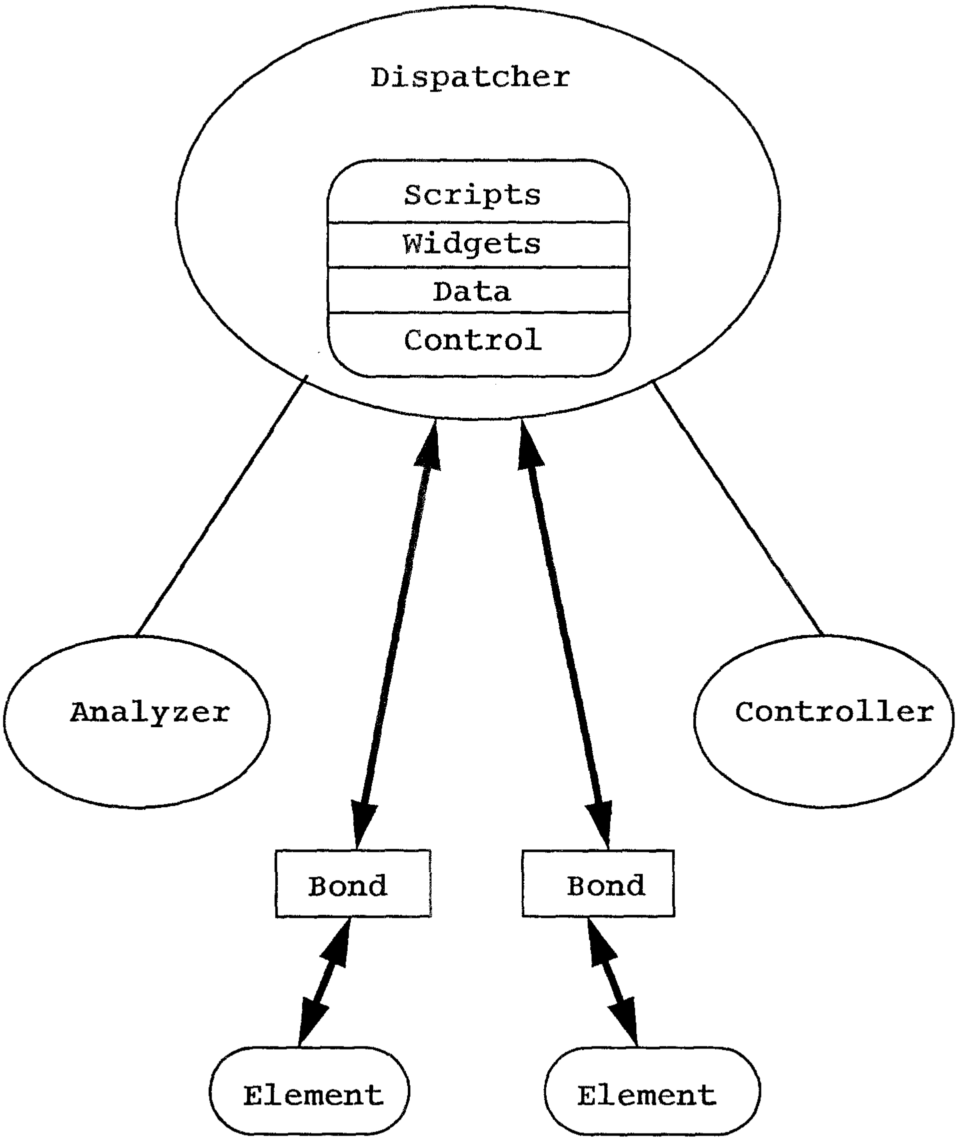
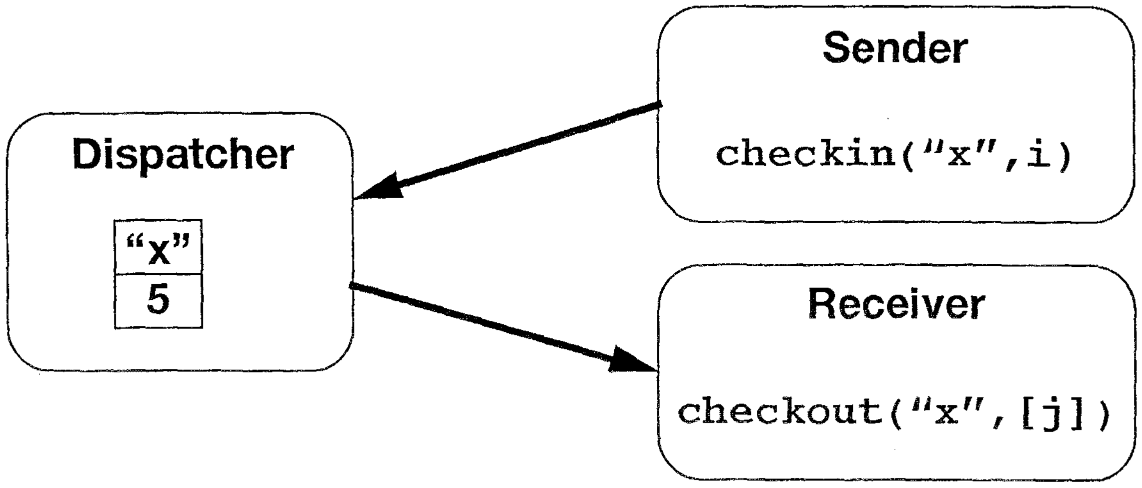


Figure 2: The SHOWME Architecture



checkin	checkout	match	transfer
formal	formal	yes?	no
formal	actual	yes	no
actual	formal	yes	yes
actual	actual	yes	-

Figure 3: Pattern Matching in Tuple Space