6-2001

Infopipes—an Abstraction for Information Flow

Jie Huang
Oregon Graduate Institute of Science & Technology

Andrew P. Black
Oregon Graduate Institute of Science & Technology

Jonathan Walpole
Oregon Graduate Institute of Science & Technology

Calton Pu
Georgia Institute of Technology - Main Campus

Infopipes—an Abstraction for Information Flow

Jie Huang **Andrew P. Black** **Jonathan Walpole** **Calton Pu**
{jiehuang, black, walpole@cse.ogi.edu} {calton@cc.gatech.edu}

Oregon Graduate Institute Georgia Institute of Technology

1. Introduction

Building Object-Oriented Distributed Systems has been facilitated by Remote Message Sending (RMS) systems like Java RMI and implementations of CORBA. However, RMS systems are designed to support request/response interactions. Streaming applications, in contrast, are characterized by high-bandwidth, long-duration communication with stringent performance requirements. Examples of streaming applications include video-on-demand, teleconferencing, on-line education, and environmental observation. These applications transfer huge amounts of data and focus on distributed information flow rather than request/response.

To simplify the task of building distributed streaming applications, we propose a new abstraction for information flow—Infopipes. Using Infopipes, information flow becomes the heart of the system, not an auxiliary mechanism that is hidden away. Systems are built by connecting pre-defined component Infopipes such as sources, sinks, buffers, filters, broadcasting pipes, and multiplexing pipes. An Infopipe has a data interface that pulls information items from the upstream Infopipes, or pushes them into the downstream Infopipes, or both. An Infopipe also has a control interface that dynamically monitors and controls the properties of the Infopipe, and hence the properties of the information flowing through it. We intend to provide property-preserving composition of Infopipes, so that the properties of the whole pipeline can be calculated from the properties of the component Infopipes in it. Quality of Service (QoS) requirements then can be analyzed and understood system-wide.

In Section 2 we discuss related technologies. In Section 3 we describe the concepts of Infopipes. We report our work on an Infopipe prototype in Section 4. Section 5 summarizes the features of Infopipes and lists some open questions.

2. Background of Infopipes

Pipes for information flow are not a new idea in computer science. In UNIX, pipes have long been used to connect processes, one of which sends a flow of information to the other. FIFOs and Streams are improved forms of pipes. These Unix mechanisms have shown that pipes are a convenient and natural abstraction for transferring information.

However, these mechanisms are not sufficient to deal with the variety of operating system platforms, network protocols, and devices in the Internet. RMS systems and similar middleware are designed to address cross-platform problems. They hide the differences between various operating systems, programming languages, networks, and processors. They encapsulate the error-prone steps of locating and accessing remote hosts, marshalling and unmarshalling. The emphasis of RMS is to make remote method invocations look like local ones. So it is well suited to request/response interactions among client/server applications. The QoS requirements of streaming applications are such that though frequently cannot tolerate the performance overhead of RMS systems [Schmidt 1992, Mungee et al. 1999].

The mismatch between streaming applications and middleware has long been addressed. The Open Software Foundation Distributed Computing Environment provides a *pipe* datatype in its RPC implementation to support the streaming of very large parameters [Fauth et al. 1991]. The Object Management Group has defined the CORBA Audio/Video Streaming Service Specification, which separates its stream establishment and control protocols from its data transfer protocols. While the establishment and control signals still pass through the Object Request Broker (ORB), applications can select the most suitable data transfer protocols for particular network environment or application requirements [Mungee et al. 1999].

This work was partially supported by DARPA/ITO under the Information Technology Expeditions, Ubiquitous Computing, Quorum, and PCES programs, by NSF award CDA-9703218, and by Intel.

All these efforts aim at efficient transfer of a large amount of data by providing raw data streaming. Infopipes support information transfer at a higher level of abstraction than the raw data. Infopipes maintain QoS properties in information flow to help meet the diverse QoS requirements of streaming applications. These properties include transport layer properties such as data rate, jitter, bandwidth and latency, and application specific properties such as frame rate and play speed for streaming video. The Infopipe abstraction hides the complexity of maintaining and managing them.

Many of the ideas in Infopipes come from our experience of building a video pipeline at the Oregon Graduate Institute [Walpole et al. 1997; Krasic et al. 1999]. We also make use of component technology and current middleware systems.

The current goal of Infopipes is to satisfy the requirements of distributed multimedia applications. The ultimate goal is to provide smart delivery of fresh information for the Infosphere project, which is ongoing at Georgia Institute of Technology and Oregon Graduate Institute [Pu 2000; Pu et al. 2001; Liu et al. 2000].

3. Infopipes

When working on the plumbing in a building, a plumber needs to study the structure of the building, to decide the path of water flow, and to calculate the parameters of pipes and fixtures. However, he does not bother to design or to make pipes. Instead, he selects from pipes and fixtures on the shelves of hardware stores. He connects the pipes and fixtures with the right sizes, lengths, and shapes. Soon a working system is ready for use. This is the way we build an information flow system using Infopipes as shown in Figure 1.

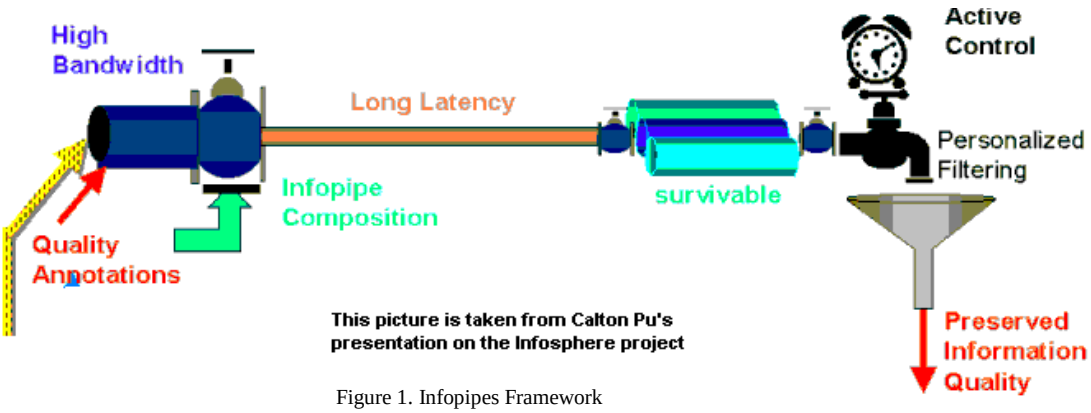

Figure 1. Infopipes Framework

3. 1 Pre-defined Infopipes

An Infopipe has zero or more in-ports and out-ports. These ports are the connectors of the Infopipe, to which the ports of other Infopipes will be joined. An Infopipe also has well-defined properties such as rate, latency, and jitter [Black 2000]. We have defined several types of Infopipes that are typical in an information flow system. Some of them are shown in Figure 2.

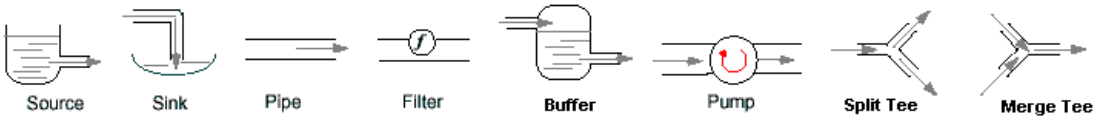

Figure 2. Some Infopipes

Sources provide the information needed to be transferred. Examples of sources are digital cameras, microphones, environmental sensors, files, web pages, and databases. *Sources* do not have in-ports.

Sinks are the destinations in an information flow system. They may be MPEG players that decompress the streamed-media and display it. They may be files or database systems that store the information. *Sinks* do not have out-ports.

Pipes transmit data. Communication links such as network connections, IPC channels, and hardware buses may be wrapped as *pipes*. *Pipes* may vary in bandwidth, latency, or rate, but they all have one in-port and one out-port.

Filters transform the information items flowing through them. The functions of *filters* include compression, decompression, labeling, delabeling, encryption, and decryption. *Filters* can also implement dropping policies. *Filters* provide computation capability in an information flow system.

Buffers provide temporary storage. They are used to smooth the jitter and the temporary differences between input rate and output rate. *Buffers* are also the right place for flow control or reordering.

Pumps add "energy" to the system. The information in many *sources* will not flow out until it is requested. Many *sinks* wait for the data to be pushed in. *Pumps* drive the information flow. Obviously, they are the critical components for controlling data rate.

An information flow system is not always a straight line. So we have many kinds of branching pipes called *tees* to construct systems with different topologies. We use *merge tees* to combine more than one stream together. For example, we can use a merge tee to combine the video stream and the audio stream of a MPEG movie. *Split tees* distribute information flows. For example, we can use a *split tee* to distribute a stereo audio stream to two speakers. *Multicast tee*s are *split tees* that send each input item to all out-ports. There are two kinds of *switching tees*. A *switching split tee* has one in-port and multiple out-ports; one out-port is selected for each input item. A *switching merge tee* has multiple in-ports and one out-port; at any time, information items from only one in-port are transmitted; the selection is based on the information item being transferred, the history, and the control information. *Tees* greatly improve the flexibility of Infopipes but increase the complexity of property preservation and transfromation.

3. 2 Polarity of Infopipes

When two Infopipes are connected, either of them may initiate the process of information transfer. So the data interfaces of Infopipes have two operations: push and pull. As shown in Figure 3, an Infopipe may call the push operation of its downstream Infopipe to send information items, or call the pull operation of its upstream Infopipe to request information items.

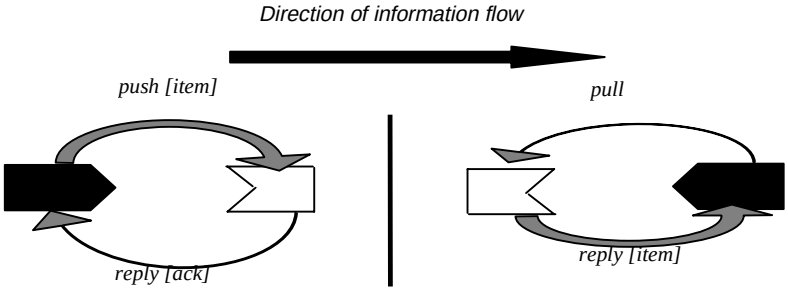

Figure 3. Push and pull operations

However, we do not want it to be the case that both the source end and the destination end initiate data transmission, or that neither of them initiate data transmission. These two cases are shown in Figure 4. In the first case, data transmission will run out of control in that the rate and the order of data items are unpredictable. In the latter case, no data will be transmitted at all.

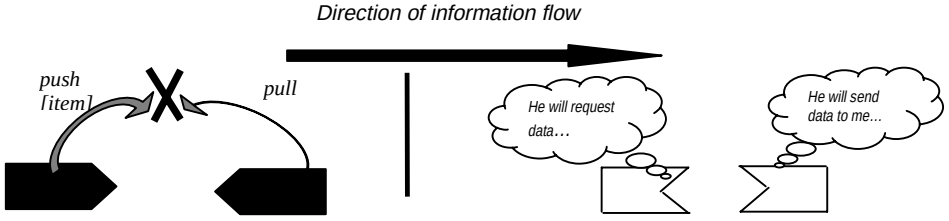

Figure 4. Incorrect connections

To check whether Infopipes can be connected, we assign a polarity to each port of an Infopipe. Ports through which an Infopipe actively sends out data or requests data are assigned positive polarities. These ports send out the push or pull messages. Ports through which an Infopipe passively receives data, or sends out data by replying to a request, are assigned negative polarities. These ports execute push or pull operations when they get the messages. Only ports with opposite polarities can be connected. This rule is shown is Figure 5. If all Infopipes are connected correctly, we are sure that information can flow through the resulting pipeline.

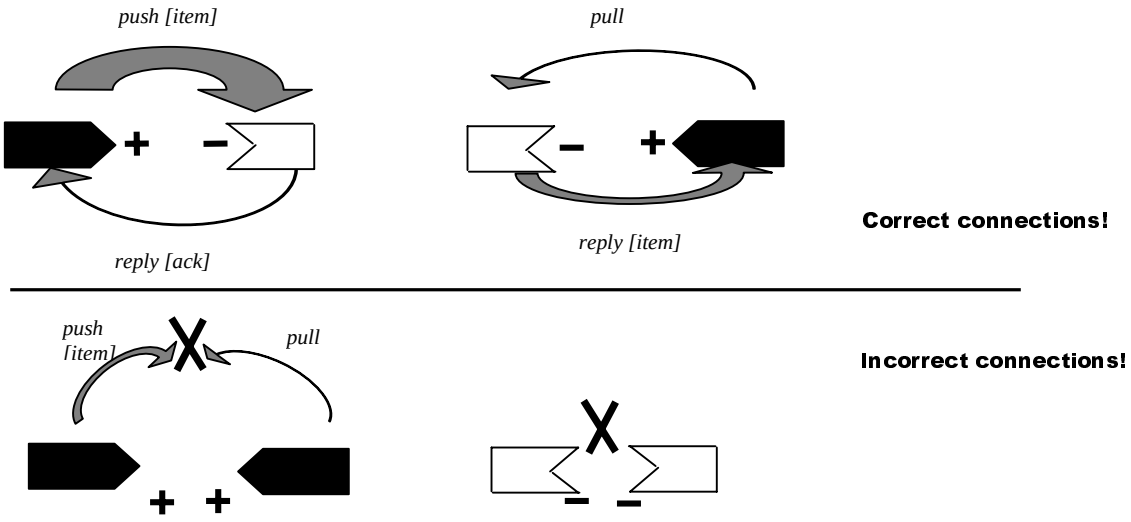

Figure 5. Polarities and connections

Let's examine the polarities for primitive Infopipes. A *pump* has a positive in-port and a positive out-port. It actively pulls data from upstream a nd pushes data downstream. A *pump* is not the information source but the energy source of a system. We write pump: $+ \rightarrow +$.

Buffers are naturally passive because they serve as storage. They have a negative in-port and a negative out-port. They are the energy consumers in the system. We write buffer: $- \rightarrow -$.

The case of *pipes* is a little bit more complex. For a water pipe, if we keep pushing water into it, the water comes out from the other end automatically. This phenomenon suggests that a *pipe* have a negative in-port and a positive out-port. But we can also pull water from the output end of a pipe, as long as its input end is in a pool. The pipe propagates the pull power and draws water from the pool. So it is also reasonable for a *pipe* to have a positive in-port and a negative out-port. *Pipes* work well in both ways. For example, a client can send data to a server through a network connection spontaneously, or it can reply to the server's request for data through the same connection. In this sense, *pipes* are polymorphic. We write pipe: $\alpha \rightarrow \overline{\alpha}$, in which α is a polarity variable, which may be either positive or negative, and $\overline{\alpha}$ denotes the opposite of α.

Figure 6 shows the polarity for Infopipes in our current design.

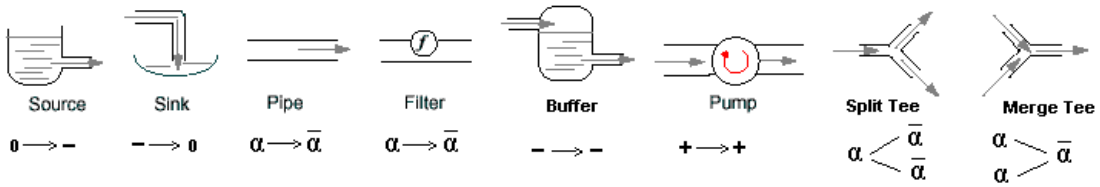

Figure 6. Polarity and Infopipes

3. 3 Composite Pipes

Suppose there are two identical bathrooms in a house. The plumber does not need to do the design twice. When he is working on the second bathroom, he can simply make a copy of the pipeline in the first one.

In Infopipes, we provide a mechanism to help to reuse a composite pipeline. If we build a group of connected Infopipes with a reusable structure and favorable properties, we can put it into a black box called a *composite pipe*. Then we can copy it and reuse it at other places as if it were a basic Infopipe.

One difficulty in copying a *composite pipe* for reuse is to decide which attribute values should be kept and which attribute values should be discarded. For example, pumping rate and buffer size should be copied if we want the copy to have the same properties. In contrast, network addresses and file names should be discarded because the copy is likely to be used in another location.

Even though we have only a limited number of basic Infopipe types, *composite pipes* allow us to construct a complex information flow system conveniently.

3. 4 Control Interfaces and Feedback Control

Different Infopipes may provide different control interfaces. Some control interfaces also depend on the information flow. Currently, we have fillLevel for *buffers* and slower and faster for *pumps*. We are investigating the properties and control information that should be maintained in Infopipes and information flows to support more control interfaces.

Once the control interfaces are defined, we can use SWiFT [Goel et al. 1999] to build software feedback controls, which allow a system to adapt to both sudden and gradual changes in environmental variables such as network congestion and CPU saturation. The basic blocks in SWiFT are feedback components, such as monitors, actuators, low pass filters, average filters, compensators, and so on. These components may be composed into more sophisticated feedback mechanisms with predictable properties and failure modes. Figure 7 shows an example of feedback control. The controller monitors the fill level of the buffer. If it is almost full, the controller tells the pump to slow down. If it is almost empty, the controller tells the pump to speed up. By using different compositions of SWiFT components and adjusting the components' parameters, the controller can implement different control policies.

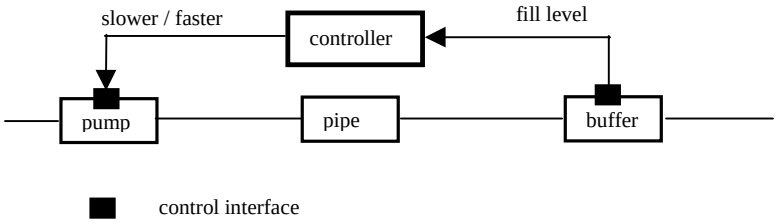

Figure 7. Control interface and feedback control

4. Infopipe Prototype

We have implemented a prototype of Infopipes in Squeak, a dialect of Smalltalk [Squeak 2001]. Each kind of Infopipe is a class in Squeak. Data items are transmitted through push operations and pull operations. Each Infopipe class has a set of named in-ports and a set of named out-ports. Infopipes are connected through their ports. Ports have a polarity according to how they are used. We implement a polarity checker to ensure that only ports with opposite polarities can be connected. We implement polarity checking using a unification algorithm to retain the polymorphism of composed Infopipes.

We provide a *netpipe* class to support connections between machines. All details of networking are encapsulated in the *netpipe* and it provides the same interfaces as a *buffer*. Connections between two Infopipes always have local interfaces. We use a remote message passing package called S2S to send the connection messages between two machines. S2S is like Java RMI for Squeak. Internally *netpipes* use UDP to transmit data. Other version of *netpipe* might use other transport protocols according to the requirements of applications.

We have built a streaming MIDI player as a demonstration of the Infopipe prototype. It has a simple feedback control mechanism similar to the controller in Figure 7.

5. Concluding Remarks and Future Work

Infopipes are a new abstraction for transferring information over the Internet. They have following features:

- focus on information flow,
- explicit support of layered QoS properties,
- support for information flow composition,
- property preserving composition, and
- feedback-based adaptation.

These features distinguish Infopipes from other systems and facilitate the timely delivery of large quantities of high quality information.

We believe that the following questions are worth further researchs:

- What QoS properties should be maintained in Infopipes and information flows? How should they be represented and managed?
- How should the properties be composed? How should they be translated at different levels?
- Develop a Domain Specific Language to describe an Infopipe system and a generator to translate the description to implementation.

Acknowledgements

This work is part of the Infosphere Project, a DARPA-funded collaborative project led by Calton Pu, Ling Liu and Karsten Schwan at Georgia Institute of Technology and Jonathan Walpole at Oregon Graduate Institute. Development and prototyping of different aspects of the Infopipe abstraction is underway in parallel at both sites. The ideas in this paper are a result of this collaboration.

Reference:

[Black 2000] Andrew P. Black. Infopipes: an Abstraction for Information Flow. Slides for the talk given at Cambridge University Computing Laboratory, England, http://www.cse.ogi.edu/~black/presentatins/.index.html#InfoPipes, October 26th 2000.

[Fauth et al. 1991] Dietmar Fauth and Chikong Shue. Remote Procedure Call: Technology, Standardization and OSF's Distributed Computing Environment. In Proceedings of *ONLINE*'91, Hamburg, Germany. Open Software Foundation.

[Goel et al. 1998] Ashvin Goel, David Steere, Calton Pu, and Jonathan Walpole. SwiFT: A Feedback Control and Dynamic Reconfiguration Toolkit. *Technical Report CSE-98-009, Oregon Graduate Institute*, June 1998.

[Krasic et al. 1999] Charles Krasic and Jonathan Walpole. QoS Scalability for Streamed Media Delivery. *CSE Technical Report CSE-99-011, Oregon Graduate Institute*, September 1999.

[Liu et al. 2000] Ling Liu, Calton Pu, Karsten Schwan, and Jonathan Walpole. InfoFilter: Supporting Quality of Service for Fresh Information Delivery. New Generation Computing Journal (Ohmsha, Ltd. and Springer-Verlag), Special issue on Advanced multimedia Content Processing, Vol.18, No.4, August, 2000

[Mungee et al. 1999] S. Mungee, N. Surendran, Y. Krishnamurthy, and D. C. Schmidt. The Design and Performance of a CORBA Audio/Video Streamign Service. *Hawaiian International Conference on System Sciences(HICSS)*, Minitrack on Multimedia DBMS and the WWW, Hawaii, January 1999.

[Pu 2000] Calton Pu. Infosphere: Smart Delivery of Fresh Information. Presentation slides. http://www.cc.gatech.edu/projects/infosphere/SBBDkeynote/ppframe.htm.

[Pu et al. 2001] Calton Pu, Karsten Schwan, and Jonathan Walpole. Infosphere Project: An Overview. Submitted to SIGMOD Record, January 2001.

[Schmidt 1992] Douglas C. Schmidt. IPC SAP: A Family of Object-Oriented Interfaces for Local and Remote Interprocess Communication. *C++ Report*. November/December 1992.

[Squeak 2001] The Squeak web site. www.squeak.org.

[Walpole et al. 1997] Jonathan Walpole, Rainer Koster, Shanwei Cen, Crispin Cowan, David Maier, Dylan McNamee, Calton Pu, David Steere, and Liujin Yu. A Player for Adaptive MPEG Video Streaming over the Internet. *Proceedings 26th Applied Imagery Pattern Recognition Workshop AIPR-97*, SPIE, Washington DC, October 1997.