January 2001

Infopipes for composing distributed information flows

Rainer Koster

Andrew P. Black

Jie Huang

Jonathan Walpole

Infopipes for Composing Distributed Information Flows

Rainer Koster
University of Kaiserslautern, koster@informatik.uni-kl.de
Andrew P. Black, Jie Huang, Jonathan Walpole
Oregon Graduate Institute, {black,jiehuang,walpole}@cse.ogi.edu
Calton Pu
Georgia Institute of Technology, calton@cc.gatech.edu

Abstract

Building applications that process information flows on existing middleware platforms is difficult, because of the variety of QoS requirements, the need for application-specific protocols, and the poor match of the commonly used abstraction of remote invocations to streaming. We propose Infopipes as an high-level abstraction for building blocks that handle information flows. The ability to query individual Infopipe elements as well as composite Infopipes for properties of supported flows enables QoS-aware configuration. Similarly to local protocol frameworks Infopipes provide a flexible infrastructure for configuring communication services from modules, but unlike protocols the abstraction uniformly includes the entire pipeline from source to sink, possibly across process and node boundaries.

1 Introduction

Interaction using common middleware platforms is based on mechanisms such as remote procedure call or remote object invocation. Procedure and method interfaces are specified in an *interface definition language* (IDL). Using this specification of a service, client and server can be developed independently of each other. Moreover, from an IDL description stubs and skeletons can be generated that handle remote communication via a standard protocol, transparently to the application developer.

This approach, however, does not support streaming and processing of distributed information flows well.

- The request-response style of interaction is built on control flowing to the server and back to the client, rather than on a continuous information flow.
- Information flows typically have timing and other non-functional requirements. Hence, *Quality of Service* (QoS) properties need to be an integral part of the abstraction. Besides, a built-in standard protocol is insufficient for supporting this variety of requirements.
- Conventional middleware builds application-specific components (client and server) on top of generated (stubs and skeletons) and system-provided (ORB, standard protocol) components. For flows, application-specific and platform-provided components may be mixed in various ways.

A flow-based application typically wants information to be transmitted from a source to a sink with specific flow properties being maintained. A variety of components may be needed for this task. Some of them are structured in pairs like protocol layers such as compression and decompression, marshaller and unmarshaller, or feedback actuator and sensor. Other elements such as filters and format converters can be placed in several positions of the pipeline. Hence, it is important to enable applications to control their own structure and to have well defined interfaces between each two stages rather than the three interfaces used by standard middelware: client-stub, standard protocol, and skeleton-server. Most of these elements are built to process specific types of information. Their functionality is neither general enough to provide them as part of the middleware platform nor specific to particular applications. Hence, there needs to be support for reusing and composing these pipeline stages.

As an example, consider a simple video pipeline from a source producing compressed data to a display. Then, the different formats require a decoder for that codec. If producer and consumer are on different nodes connected by a best-effort network, a feedback mechanism should be used to control, which data is dropped, rather than incurring arbitrary dropping in the network. The feedback and decoding elements are likely to be specific to the

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

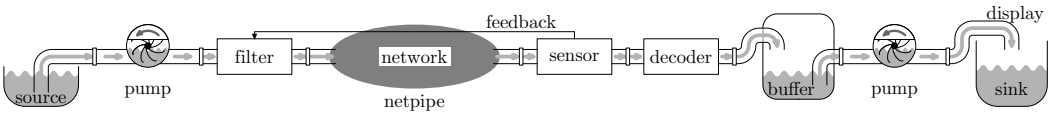

Figure 1: Infopipe Example

given flow format, but can be used in video on demand, video conferencing, or surveillance applications.

The Infopipe abstraction simplifies the task of building distributed streaming applications by providing basic elements such as pipes, filters, buffers, and pumps [1, 9]. Each element specifies the properties of the flows it can support, including data formats and QoS parameters. When stages of a pipeline are connected flow properties for the composite can be derived, facilitating the composition of larger building blocks and incremental pipeline setup.

Section 2 describes the Infopipe abstraction. Section 3 discusses an approach to support application-controlled pipeline setup. Related work is summarized in Section 4 before the conclusions in Section 5.

2 Infopipes

Infopipes model pipeline elements for information flow analogously to plumbing for water flow. The goal is supporting a similarly simple composition of pipelines from elements.

The most common elements have one input and one output. Such pipes can just *transport* information, *filter* certain information items, or *transform* the information. *Buffers* provide temporary storage and remove rate fluctuations that cause jitter. There are *pumps* to keep the information flowing, pulling items from upstream and pushing them downstream. Hence, pumps have two active ends and buffers have two passive ones, while filters and transformers have two ends of opposite polarity but can typically be used in either push or pull mode [1]. *Sources* and *sinks* have only one end, and can be either active or passive.

More complex pieces have more ports. Examples are *tees* for splitting and merging information flows. Splitting includes splitting an information item into parts that are sent different ways, copying items to each output (multicast), and selecting an output for each item (routing). Merging can combine items from different sources into one item or pass on information to one output in the order it arrives at any input.

In combining elements of a pipeline it is important to check the compatibilty of supported flows and to evaluate the characteristics of the composite Infopipe. From each basic or composite Infopipe a *Typespec* of supported flows can be queried. These types include supported formats of data items, interaction properties such as the capability of operating in push or pull mode, and ranges of QoS parameters that can be handled.

To integrate different transport protocols into the Infopipe framework, they can be encapsulated in *netpipes*. These netpipes support plain data flows and may handle low-level properties such as bandwidth and latency. Marshalling filters on either side translate the raw data flow to a higher-level information flow and vice-versa. These elements also encapsulate the QoS mapping translation between netpipe properties and information flow specific properties.

Figure 1 shows the video pipeline used as an example in Section 1. At the producer side, frames are pumped through a filter into a netpipe encapsulating a best-effort transport protocol. The filter drops frames controlled by a feedback mechanism using a sensor on the consumer side. After decoding, the frames are buffered to reduce jitter. A second, timer-controlled pump finally releases the frames to the display sink.

The Infopipe abstraction has emerged from our experience building continuous media applications. Currently we are building a middleware framework based on these concepts. On top of this platform we are going to reimplement our video pipelines to facilitate further development.

3 Distributed Setup

Due to the diversity of application-specific QoS requirements and trade-offs, the Infopipe framework does not try to build pipelines automatically from declarative requirements but lets the application itself control its structure. For setting up protocol stacks in this procedural way hierarchical blueprints have been proposed [5]. These blueprints can contain alternative configurations that may be chosen depending on the availability of required sub-components and resources. Moreover, they are not necessarily complete, but may be mixed with declaratively configured parts.

This hybrid approach can be used for Infopipes. While the right choice of compression algorithms, feedback mechanisms, or underlying transport protocols is likely to be application-specific and com-

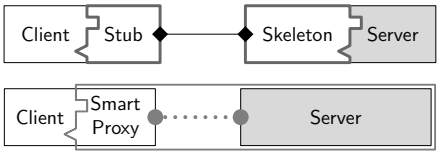

Figure 2: Smart Proxies

ponents should be explicitly selected, the flow properties can also be used for partial declarative configurations. If data needs to be converted from one format to another, a transformation can be chosen automatically as in some exisiting streaming frameworks [6,10]. The classical case of marshalling, that is the conversion between a network-packet representation and an in-memory representation,[1] falls into this categorie. In this case, the transformation component can even be automatically generated.

For Infopipes, there also needs to be an interface between consumer and producer applications, similar to the need for a well-defined interface between clients and servers. For application-level protocols such as FTP the service interface is defined in terms of commands sent over a connection, and for conventional message-based middleware it is defined in IDL. The former interface is low-level and between the hosts, the latter is high-level and identical on either side. For an Infopipe consisting of potentially many stages, it is more difficult to locate the interface that conceptually separates consumer and producer. The low-level transport-protocol interface would not be a good choice, because a consumer and a producer would only be compatible if they happen to use the same protocol, even if they handle the same type of flow on a higher level. Choosing a high-level interface locates it necessarily on either side of the network. Producers or consumers could be in charge of configuring the remote flow.

For realising this high-level approach in CORBA with additional transport protocols, dynamically loadable *Smart Proxies* have been proposed [3, 4]. The server logically extends to the client node and controls the network part of the pipeline as shown in Figure 2. At connection setup, the server chooses a communication mechanism based on information about the available resources. A video server, for instance, could use shared memory if it happens to be on the same node as the client, a compression mechanism and UDP across the Internet, or raw Ethernet on a dedicated LAN. It then transmits the code for a Smart Proxy to the client implementing the respective communication endpoint functionality. In this way, application specific remote communication can be used without making the network protocol the actual service interface. In that case, all client applications would have to implement all protocols that are used by any servers they may ever connect to. With Smart Proxies, the service interface can still be described on a high level in an IDL. Client applications are programmed accessing this interface as if the server were on the same machine, but actually communicate with the proxy.

The same idea can be applied to Infopipes in a generalized way. Since there are high-level interfaces between all elements of a pipeline the granularity of composition can be finer. It is not necessary to send monolithic proxies implementing everything between the consumer interface and the network. It is rather possible to use standard pipeline elements that may be available on the consumer side for composing the required functionality. It may be sufficient to send a proxy blueprint as discussed above, or to send small specialized elements in addition.

The consumer could be put in charge of the network, too. In this way, application-specific adaptation policies could be uploaded to the producer, for instance. It is also possible to decouple responsibilities even further and let a third party control the network pipeline. Then both consumers and producers can benefit from various transport pipelines that can be developed and improved independently. A specialized transportation service needs to send configurations and potentially code to either side.

4 Related Work

Some efforts aim at integrating streaming services with middleware platforms based on remote object invocations such as CORBA. The CORBA telecoms specification defines stream management interfaces, but not the data transmission. Extensions to CORBA such as TAO's pluggable protocol framework allow the efficient implementation of audio and video applications [7].

The QuO architecture [12] complements the IDL descriptions with specifications of QoS parameters and adaptive behavior in special languages. From these declarative descriptions so called delegates are generated and linked to the client application in a similar way as stubs are generated from the IDL. While this approach maintains platform and language independence, its flexibility is limited by the capabilities of the generators and the static linking of delegates.

In the Jini environment [13] client-server communication is encapsulated in proxies that are shipped at run time. Since Jini is based on Java, it inherits the advantages of security, ease of code shipping, and platform independence, as well as the drawbacks of being restricted to one language and the

[1] There are several in-memory representations in a heterogeneous environment.

potential performance penalties and unpredictability of a virtual machine.

Blair et al. proposed a procedural approach to reflection as a general design principle for a middleware architecture [2].

Modular protocol frameworks such as Ensemble [11] or Da CaPo [8] support the composition and reconfiguration of protocol stacks from modules. Both provide mechanisms to check the usability of configurations and use heuristics to build the stacks. Unlike these frameworks for local protocols, Infopipes use a uniform abstraction for handling information flow from source to sink, possibly across several network nodes. Moreover, the application controls the pipeline setup.

5 Conclusions

Infopipes provide a framework for building pipelines from elements for processing information flows. This abstraction uniformly extends from source to sink. The application controls the setup of the pipelines, configuring their behavior based on QoS parameters and other properties exposed by the elements.

Starting from two prototype implementations in Smalltalk and C++ exploring the general idea and the threading support respectively, we are extending the supported functionality by the distributed setup described above, resource reservations, and feedback mechanisms. As a test case, we are building a video streaming application on top of the framework.

References

[1] Andrew P. Black and Jonathan Walpole. Aspects of information flow. In *ECOOP 2000 Workshop on Aspects and Dimensions of Concerns*, 2000.

[2] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas. An architecture for next-generation middleware. In *International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware)*, pages 191–206. IFIP, September 1998.

[3] R. Koster and T. Kramp. Loadable smart proxies and native-code shipping for CORBA. In *Proceedings of the Third International Conference on Trends towards a Universial Service Market (USM)*. IFIP/GI, Springer, September 2000.

[4] R. Koster and T. Kramp. Structuring qos-supporting services with smart proxies. In *Proceedings of the Second International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware), LNCS 1795*, pages 273–288. IFIP/ACM, Springer, April 2000.

[5] T. Kramp and R. Koster. Descriptive-procedural configuration of communication bindings. In *Proceedings of the International Conference on Multimedia and Expo (ICME)*. IEEE, August 2000.

[6] Mircosoft. DirectX 8.0: DirectShow overview. `http://msdn.microsoft.com/library/psdk/directx/dx8_c/ds/0view/about_dshow.htm`, January 2001.

[7] S. Mungee, N. Surendran, and D. C. Schmidt. The design and performance of a CORBA audio/video streaming service. In *HICSS-32 International Conference on System Sciences, minitrack on Multimedia DBMS and WWW*, January 1999.

[8] T. Plagemann and B. Plattner. CoRA: A heuristic for protocol configuration and resource allocation. In *Proceedings of the Workshop on Protocols for High-Speed Networks*. IFIP, August 1994.

[9] C. Pu, K. Schwan, and J. Walpole. Infosphere project: System support for information flow applications. *ACM SIGMOD Record*, 30(1), March 2001.

[10] Wim Taymans. GStreamer application development manual. `http://www.gstreamer.net/documentation.shtml`, January 2001.

[11] R. van Renesse, K. Birman, M. Hayden, A. Vaysburd, and D. Karr. Building adaptive systems using Ensemble. Technical Report TR97-1638, Computer Science Department, Cornell University, 1997.

[12] R. Vanegas, J. A. Zinky, J. P. Loyall, D. A. Karr, R. E. Schantz, and D. E. Bakken. QuO's runtime support for quality of service in distributed objects. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*. Springer Verlag, September 1998.

[13] J. Waldo. The Jini architecture for network-centered computing. *Communications of the ACM*, 42(7):76–82, July 1999.