

Infopipes: Concepts and ISG Implementation

Galen S. Swint, Calton Pu, *Senior Member, IEEE*, and Koichi Moriyama

Abstract— We describe **Infopipes**, a distributed computational and communications abstraction for information flow applications and I/O intensive distributed real-time embedded (DRE) systems. Infopipes are specified by the syntax, semantics, and quality of service requirements for information flows. Software tools generate executable code from the specification. We explain a DRE scenario and then provide a microbenchmark comparison of generated Infopipe code to standard, hand-written TCP code. Measurements show that Infopipe-generated code has the same execution overhead as the manually written original version.

Index Terms—Software quality, Software tools, System software.

I. INTRODUCTION

One of the fundamental functions of operating systems (OS) is to provide a higher level of programming abstraction on top of hardware to application programmers. More generally, an important aspect of OS research is to create and provide increasingly higher levels of programming abstraction on top of existing abstractions. Remote Procedure Call (RPC) [1] is a successful example of such abstraction creation on top of messages, particularly for programmers of distributed client/server applications.

RPC, despite its success, has proven to be less than perfect for some applications. The primary reason for this is that it abstracts away too much information which the application often needs to run "correctly." This often results in the application developers re-writing code to discover the properties hidden by the RPC call. For instance, a streaming media application has distinct timing requirements, but an RPC call, by itself, contains no information about timing. Consequently, the writer of a streaming media application

must add timers to his or her code to recover the "lost" timing information.

The Infopipe concept, [2], [3], [4], [5], is a high-level abstraction to support information-flow applications. Example applications include data streaming and filtering [4], building sensor networks, e-commerce transactions and multimedia streaming [5].

Infopipes are being developed with the support of the DARPA PCES (Program Composition for Embedded Systems) project. With that project, our experiences have shown that Infopipes can reduce the effort and code required to develop connections between the embedded imaging system of the UAV and the receiver of the image stream. Furthermore, Infopipes can employ various data management techniques to maximize communication quality.

The focus of our work has been the implementation of a software tool – the Infopipe Stub Generator (ISG), descriptive formats for Infopipes – the Infopipe Specification Language (ISL), and a suite of Infopipe implementations. From a top-down perspective, our software tools consist of a series of translators that successively creates appropriate abstract machine code at lower levels by refining the specification and code from the previous higher level abstraction. The descriptive formats we created are intended to capture the geometry of information-flow applications and describe certain semantics pertaining to them. Finally, comparing our implementation of generated Infopipe code to hand written versions for two representative applications, we observe minimal added overhead.

The rest of the paper is organized as follows. Section II summarizes the Infopipe abstraction and our implementation. Section III describes the experimental evaluation results. Section IV summarizes related work, Section V describes our current research, and Section VI concludes the paper.

II. INFOPIPES

A. Motivation

Remote procedure call (RPC) is a well-established mechanism for constructing distributed systems and applications, and a considerable amount of distributed systems research has centered on it. The widespread use and acceptance of RPC has led to the development of higher-level architectural models for distributed system construction. For example, it is a cornerstone for models such as client/server (including Internet protocols like HTTP), DCOM, and CORBA. The client/server model is widely considered to be a good choice for building practical distributed applications,

Manuscript received January 14, 2004. This work was done as part of the Infosphere and Embedded Infopipes projects, funded by DARPA through the Information Technology Expeditions, Ubiquitous Computing, Quorum, and PCES programs. The research was also partially funded by NSF's CISE directorate, through the ANIR and CCR divisions as well as the ITR program. In addition, the research was partially funded by Intel.

G. S. Swint is a Ph.D. graduate student in the College of Computing at the Georgia Institute of Technology, Atlanta, GA 30332-0280 USA (phone: 404-385-2585, e-mail: swintgs@acm.org)

C. Pu is a Professor, holds the John P. Imlay Chair in Software, and is the Co-Director for the Center for Experimental Research in Computer Systems in the College of Computing at the Georgia Institute of Technology, Atlanta, GA 30332-0280 USA (e-mail: calton@cc.gatech.edu).

K. Moriyama is an engineer with Sony Corporation of Japan. His work was done during an extended visit to Georgia Tech.

particularly those using computation or backend database services.

On the other hand, while these models have proven successful in the construction of many distributed systems, RPC and message passing libraries offer limited support for information-driven applications because their natural operation does not reflect RPC's request-response style. One example is bulk data transfers in which large numbers of requests and responses may be generated to cope with network reliability [6]. Another example is when information flows have quality of service (QoS) requirements, then certain elements of distribution transparency – an oft-cited advantage of RPC – can cause more problems than they solve. Various solutions to these shortcomings have been proposed. For example, the A/V Streams specification was appended to CORBA to address the deficiency of RPC in handling multimedia flows.

Several emerging classes of distributed applications are inherently information-driven. Instead of occasionally dispatching remote computations or using remote services, such information-driven systems tend to transfer and process streams of information continuously (e.g., Continual Queries [7], [8]). Applications such as electronic commerce combine heavy-duty information processing (e.g., the discovery and shopping phase involves querying a large amount of data from a variety of data sources [9]) with occasional remote computation (e.g., buying and updating credit card accounts as well as inventory databases). Even static web pages can conceal an information-driven flow of information since a single request for a page frequently generates both multiple requests to sites' backend software and also more HTTP requests by the client as it renders a web page.

We have proposed that an appropriate programming paradigm for information-driven applications should embrace information flow as a core abstraction. Unsurprisingly, there are already concrete examples of existing information flow software. For example, in UNIX combining filters yields a pipeline which is a precursor of the Infopipe programming style. This abstraction aims to offer the following advantages over RPC: first, data parallelism among flows should be naturally supported; second, the specification and preservation of QoS properties should be included; and third, the implementation should scale with the application. We emphasize this new abstraction is intended to complement RPC — not to replace it. In true client/server applications, RPC is still the natural solution

Like RPC, Infopipes raise the level of abstraction for distributed systems programming and offer certain kinds of distribution transparency. Conceptually, an Infopipe is a mapping that transforms information units from its input domain to the output range between its consumer and producer ends. Furthermore, Infopipes go beyond the datatypes of RPC and specify the syntax, semantics, and quality of service (QoS) properties which are collected into a structure called a *typespec*. Toolkits are then needed to aid developers in managing and constructing Infopipes, managing their associated typespecs, and incorporating the

Infopipes into target applications which are increasing in complexity and heterogeneity.

B. Distributed Real-time Embedded Systems

Traditional embedded systems carry the connotation of being closed and self-sufficient. New sensor technology combined with high bandwidth wireless networking brings new applications that change the closed nature of embedded systems. These new embedded systems are termed distributed real-time embedded (DRE) systems and differ from traditional closed embedded systems in two important ways: first, these systems are I/O-intensive, e.g., sensors that primarily generate and pass on information streams. Second, they cooperate with the rest of a networked system, instead of being isolated and self-sufficient.

Traditional embedded systems have clearly defined functionality and limited (or no) interactions with the environment. Although they are less extensible than general-purpose computers, due to their simplicity embedded systems exhibit good systemic properties such as predictable performance, scalability with respect to well-defined workloads, and security. Guaranteeing these systemic properties is the very reason for the existence of embedded systems, since that is their main advantage when compared to general purpose computers.

Infopipes are designed to manage the new part of DRE systems: I/O-intensive information flow and cooperation with the rest of the networked world. Infopipes use an explicit definition of information flow syntax, semantics, and systemic properties to generate the appropriate software on behalf of application designers and implementers.

Examples of DRE systems include MEMS-based sensor networks and PDA-based real-time decision support systems. These I/O-intensive systems can either generate or consume a significant amount of information. Typically, the information is generated by sensors and processed by the DRE system into a digested form. DRE systems also frequently require cooperation and communication with other systems through a network. The quantity and variety of I/O and interactions make DRE systems an integral part of a larger distributed system such as the Internet.

C. Lessons from RPC

Even though RPC may not be a suitable abstraction for many applications, it has still contributed important tools for building distributed applications. Two of the most important are the Interface Description Language (IDL) and the RPC stub generator.

The IDL provides an easy way to specify the connection between two computation processes – the client and the server. It abstracts the connection and data typing and frees the programmer from the necessity of uncovering architecture-specific and even language-specific characteristics such as byte ordering or floating point representation. Given an IDL description, an RPC stub generator then implements the implicit functionality. Using it enhances program correctness and shortens development cycles, not just because it obviates the need for a developer to

write large amounts of code, but because the code it generates has already been tested and is far more likely to be "correct." Therefore, it greatly reduces both syntactic and logical errors appearing in communication code which in turn results in a great time reduction for application development.

Communication between processing stages in information flow applications inherently requires a large amount of communication code. Potentially, the code for each communication link must be tailored to the specific type of data exchanged. This code can consume a large amount of initial development time and debugging. Later changes are potentially difficult, as even a simple datatype change may require changes in several program modules. Aside from data marshalling and unmarshalling problems, developers must also contend with finding services, and then creating and maintaining links to them. For an Infopipe that processes data in n serial stages, there are at least $2^{*(n-1)}$ communication stubs. Any change in the data description carried over a single link, furthermore, presages a change in data marshalling code in two communication interfaces. Of course, some of these changes may require only a small change, as adding a simple extra integer to be propagated. Others may have a moderately more complex requirement, such as supporting a dynamic array. Then, there are changes which require extensive code writing and testing, as adding encryption or reliability code. Finally, systems built in this way are not easily portable to new communication protocols, computing platforms, or languages, and due to the nature of their development, are likely to become brittle over time as more features are added and each communication link loses coherency with the whole system.

D. Toolkit

To address the drawbacks found in RPC, we developed a toolkit to describe Infopipes and a tool to implement those descriptions. Our approach to making Infopipes language and system independent has two parts and parallels the IDL and stub generator of RPC.

With the first part, we define a generic interface for infopipe specification which can be compiled into an intermediate data format. This functionality is equivalent to the RPC IDL. The current version of Infopipe Specification Language (ISL) is dubbed Spi.

The second part is a translator and set of templates, the Infopipe Stub Generator or ISG, which consume the intermediate format generated by an ISL and produce compilable source code in a fashion analogous to an RPC stub generator. The stub generator hides the technical difficulties of marshalling and unmarshalling data and manipulating system-specific mechanisms for property enforcement. By adopting this approach we shield the application developer from the complexity of heterogeneous operating systems, differing hardware platforms, and the translation from language-level abstractions to underlying message-based implementations. For a more extensive discussion of the architecture and implementation of the toolkit, see [10].

By adopting this two-part approach we can shield the application developer from the complexity of heterogeneous operating systems, hardware, and languages by providing a uniform high-level abstraction of the information flow application. The abstraction then provides two important benefits to the programmer: first, a developer can easily move to a new underlying implementation as circumstances dictate since the abstraction encodes semantics about the application and not just implementation code. A second benefit is that the programmer can use multiple communication layers and connect together pipes written in multiple languages. In fact, a single pipe may use different underlying communication layers for each incoming or outgoing connection when connections to legacy software are involved. For instance, a pipe may receive data via a socket as its input, but it may later output that data using a second communication protocol such as HTTP. This type of situation is actually quite common, as the three-tiered architecture common for building web applications often uses HTTP for client/web server communication, and at least one other protocol (such as SQL via JDBC) for communication with backend machines.

By separating the ISL and the ISG, we achieve, again, an additional level of flexibility. Using an intermediate format as a connecting element, we can actually use several variant ISLs as input for generating an Infopipe. We developed Spi as a prototype language for describing Infopipes, and we also have developed a prototype graphical editing framework. Furthermore, we have the opportunity to incorporate other flow-based languages such as Spidle [11] which will allow us to provide their capabilities to an Infopipe developer. To achieve this logical separation, we decided to create a standard intermediate format based on XML (the XIP). This way, the second step (the actual code generation) can be developed in parallel to the design and evolution of variant Infopipe Specification Languages. Again, more information about XIP is available in [10].

III. EXPERIMENTAL EVALUATION

Research and development for Infopipes have centered on the Multi-UAV Open Experimental Platform. In this scenario, several Unmanned Aerial Vehicles (UAVs) transmit images for analysis by command and control and dissemination to other combat units. Communication between a UAV source and an image's ultimate destination may include one or more wireless connections that may suffer from high data loss rates, low-bandwidth, and competition from other information flows. With those challenges, Infopipes must manage the communication and still deliver the information in a consistent and timely fashion. Furthermore, when multiple UAVs are involved, the Infopipes must provide for the management of multiple information streams, too, with respect to one another. Again, the Infopipes must satisfy timeliness constraints.

We executed experiments on the latest stable version of the code generator to evaluate the overhead of using Infopipes. The hardware used in the experiments is a pair of Dell dual-CPU workstations with Pentium III (800MHz, 512MB,

256KB cache) running RedHat Linux. The two machines are connected through a lightly loaded 100Mb Ethernet and share the same file system, and the benchmarks are compiled with gcc version 3.1.

For the first experiment, we evaluated the latency between two Infopipes and compared this to the latency between two processes communicating via TCP sockets. Second, we perform another experiment that compares the data throughput between two Infopipes as compared to the data throughput of two processes communicating via a standard TCP connection. Each experiment was run using three different data packets. Small packets were single, 4-byte integers. This test stresses the overhead of the system. The large packets test involved sending an array of 1000 integers which resulted in a large packet size of 4096KB which highlights any performance dependencies on data size. Finally, the third test was to use a mixed packet which contained three different types of data in a 152 byte packet: an array of 100 char bytes, 10 4-byte floats, and 3 4-byte integers to evaluate how the software handles mixed types of data in the same packet. Each number reported is an average of 100 trials.

The first microbenchmark measures the latency experienced in microseconds. Note that the Infopipes only incur minimal extra latency. For each trial, small packet transfer time was the average for 100,000 consecutive transmissions. Each trial for the large packet and mixed packets was 10,000 consecutive transmissions. Table I summarizes the results.

TABLE I
LATENCY COMPARISON

	TCP Sockets (μ s)	Infopipe (μ s)
Small	62	62
Large	307	314
Mixed	76	78

In the second benchmark, we measure the throughput by sending 100,000 small packets and measuring the total time or 10,000 large or mixed packets and measuring the total time. Again, we see that Infopipes do not incur significant overhead when compared to the base socket implementation. Table II summarizes the results of our throughput comparison.

TABLE II
THROUGHPUT COMPARISON

	TCP Sockets (KB/s)	TCP Sockets (KB/s)
Small	31.8	32.3
Large	6612	6654
Mixed	978	972

IV. RELATED WORK

Remote Procedure Call (RPC) [1] is the basic abstraction for client/server software. By raising the level of abstraction, RPC facilitated the programming of distributed client/server applications. For example, RPC automates the marshalling

and unmarshalling of procedure parameters, a tedious and maintenance-heavy process. Despite its usefulness, RPC provides limited support for information flow applications such as data streaming, digital libraries, and electronic commerce. To remedy these problems, extensions of RPC such as Remote Pipes [6] were proposed to support bulk data transfers and sending of incremental results.

Also, as mentioned earlier, information flow programming is common in the UNIX environment when processes are combined using UNIX pipes.

Other projects that emphasize streaming media include Spidle [11], and StreamIt [12]. Spidle is a DSL developed by at the University of Bordeaux which allows existing libraries for processing streaming media to be abstracted in a flexible reusable fashion and then provide powerful optimization techniques. However, Spidle differs from Infopipes in presuming synchronous local communication, as opposed to the Infopipe model of distributed asynchronous communication. StreamIt, is also a DSL, however, to-date it has concentrated on program analysis and development of a compiler for the Raw processor.

V. CURRENT WORK

One of the important qualities of our code generation system is its high degree of flexibility. Because of the intermediate format we have chosen for the data packaging between the language compiler and the code generator, we are able to simultaneously support multiple languages of Infopipe implementation (Java, C, C++) and multiple communication packages for each language (e.g. for C++ we support Berkeley TCP and UDP sockets, in-process communication, and Linux inter-process communication, or IPC). Communication packages need not be byte-oriented as sockets are. For instance, we support a publish-subscribe middleware, ECho [13], as a communication layer binding for C. The flexibility of the toolkit has also allowed us to support several different avenues of simultaneous research efforts.

Currently, we are involved in investigations of new technologies for Infopipes. Our work in specialization ([14], [15], [16]) shows promise for optimizing connections between Infopipes and reducing their resource usage by automatically selecting connection code based on the environment in which the pipes are currently executing. For instance, pipes do not need socket connection code when both are running on the same machine. In such a case IPC may be a much better fit incurring lower overhead. By removing the socket code, we can effectively reduce the footprint of the application with no penalty in terms of functionality. Our specialization efforts center on language constructions that concisely and clearly describe these situations as well as high-performance implementations.

A second area we are investigating is adaptation to meet quality of service goals. This effort centers on pipes adjusting communication by imposing filters in between sending and receiving ends of two connected Infopipes. A filter is most easily described as code which reduces the bandwidth

necessary for two Infopipes to communicate. A typical scenario for this would be two pipes that coordinate the installation of image compression code, such as a JPEG or GZIP filter, when bandwidth is not available to send uncompressed images at the desired rate. Important in this effort is codifying when each filter should be applied and in what order. For instance, the GZIP filter may be tried first because it provides the most information, and the JPEG filter would supplant that if network bandwidth dropped further. Research on providing these features involves creating language definition for quality of service descriptions as well as a library of filters. This effort is part of the PCES Multi-UAV scenario which also includes QoS research projects by other groups such as QuO [17] and AQuA [18].

Our most recent exploration has involved incorporating "separation of concerns" concepts from Aspect Oriented Programming into the code generator itself. This allows a developer to actually modify the generated code in an explicit and a controlled fashion at generation time rather than "hacking" the generated stubs to achieve some new functionality. Furthermore, our AOP weaver allows aspects to compose on top of each other so that developers may build a library of aspects for use in common situations. Using our AOP weaver in an experiment that required automatic coordinated CPU control, we discovered that we could direct the generation of over 50% of the final code which otherwise would require hand-adapting of the Non-AOP generated output. The PCES project also includes AOP experimentation in interpreted as well as compiled, stubs or parts of stubs [19].

VI. CONCLUSION

In this paper, we describe Infopipes and an overview of our approach to creating an Infopipes implementation. We then present some microbenchmarks that demonstrate that using Infopipes does not incur significant overhead over hand-coded implementations. Also, we explain our current efforts regarding research into expanding and updating Infopipe capabilities. In all research efforts, we emphasize improving the overall quality of the generated code as well as standardizing language APIs for Infopipes.

These experiments show the promise of our implementation. We are moving forward with defining a language API for Infopipes, the addition of QoS support and the application of program specialization techniques to improve the performance of generated code as well as efforts concerning AOP. Future plans include collaboration with the Compose group of the University of Bordeaux for integration of Infopipes with Spidle.

VII. ACKNOWLEDGEMENTS

Preetesh Banthia ran the benchmarks presented in this paper, and Younggyun Koh and Wenchang Yan assisted with implementing the toolkit. In addition, we have benefited from interaction and dialog with Charles Consel of ENSEIRB, U. of Bordeaux, France; Jonathan Walpole of the Oregon

Graduate Institute; and also Ling Liu and Karsten Schwan here at the Georgia Institute of Technology.

REFERENCES

- [1] A. Birrell and B. Nelson, "Implementing Remote Procedure Calls", in *ACM Transactions on Computer Systems*, Vol. 2, No. 1, February 1984, Pages 39-59. Also appeared in Proceedings of SOSP'83.
- [2] A. Black, J. Huang, R. Koster, J. Walpole, and C. Pu, "Infopipes: an Abstraction for Multimedia Streaming", in *ACM Multimedia Systems Journal*. To appear in 2002.
- [3] R. Koster, A. Black, J. Huang, J. Walpole and C. Pu, "Infopipes for Composing Distributed Information Flows". In the *Proceedings of the ACM Multimedia Workshop on Multimedia Middleware*, Ottawa, Canada, October 2001.
- [4] L. Liu, C. Pu, K. Schwan and J. 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.
- [5] C. Pu, K. Schwan, and J. Walpole, "Infosphere Project: System Support for Information Flow Applications", in *ACM SIGMOD Record*, Volume 30, Number 1, pp 25-34, (March 2001).
- [6] D. Gifford and N. Glasser, "Remote Pipes and Procedures for Efficient Distributed Communication", in *ACM Transactions on Computer Systems*, Vol. 6, No. 3, August 1988, Pages 258-283.
- [7] L. Liu, C. Pu, W. Tang, and W. Han, "Conquer: A Continual Query System for Update Monitoring in the WWW", *International Journal of Computer Systems, Science and Engineering*. To appear in the Special issue on Web Semantics, 1999.
- [8] L. Liu, C. Pu, and W. Tang, "Continual Queries for Internet Scale Event-Driven Information Delivery", *IEEE Transactions on Knowledge and Data Engineering*, Special issue on Web Technologies, Vol. 11, No. 4, July/August 1999.
- [9] D. Steere, A. Baptista, D. McNamee, C. Pu, and J. Walpole, "Research Challenges in Environmental Observation and Forecasting Systems", *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom'00)*, Boston, August 2000.
- [10] Pu, Calton, Galen Swint, Charles Consel, Younggyun Koh, Ling Liu, Koichi Moriyama, Jonathan Walpole, Wenchang Yan. "Implementing Infopipes: The SIP/XIP Experiment." Georgia Tech Research Report GIT-CC-02-31. Available <http://www.cc.gatech.edu/projects/infosphere/papers/GIT-CC-02-31.ps>
- [11] Charles Consel, Hedi Hamdi, Laurent Réveillère, Lenin Singaravelu, Haiyan Yu, Calton Pu. "Spidle : A DSL Approach to Specifying Streaming Applications". LABRI Research Report 1282-02.
- [12] William Thies, Michal Karczmarek, and Saman Amarasinghe. "StreamIt: A Language for Streaming Applications." In *Proceedings of the 2002 International Conference on Compiler Construction..* Grenoble, France, April, 2002
- [13] Greg Eisenhauer, Fabian Bustamante and Karsten Schwan. "Event Services for High Performance Computing," *Proceedings of High Performance Distributed Computing (HPDC-2000)*
- [14] D. McNamee, J. Walpole, C. Pu, C. Cowan, C. Krasic, A. Goel, P. Wagle, C. Consel, G. Muller, and R. Marlet, "Specialization Tools and Techniques for Systematic Optimization of System Software", *ACM Transactions on Computer Systems*, Vol. 19, No. 2, May 2001, pp 217-251.
- [15] G. Muller, R. Marlet, E.N. Volanschi, C. Consel, C. Pu and A. Goel, "Fast, Optimized Sun RPC using Automatic Program Specialization", *Proceedings of the 1998 International Conference on Distributed Computing Systems*, Amsterdam, May 1998.
- [16] C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole and K. Zhang, "Optimistic Incremental Specialization: Streamlining a Commercial Operating System", *Proceedings of the Fifteenth Symposium on Operating Systems Principles (SOSP'95)*, Colorado, December 1995.
- [17] J.P. Loyall, D.E. Bakken, R.E. Schantz, J.A. Zinky, D.A. Karr, R. Vanegas, and K.R. Anderson, "QoS Aspect Languages and Their Runtime Integration. *Lecture Notes in Computer Science*, Vol. 1511, Springer-Verlag. Proceedings of the Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR98), 28-30 May 1998, Pittsburgh, Pennsylvania.
- [18] M. Seri, T. Courtney, M. Cukier, and W.H. Sanders. An Overview of the AQuA Gateway. *Proceedings of the 1st Workshop on The ACE ORB (TAO)*, St. Louis, MO, August 5-6, 2001.

- [19] A.S. Gokhale, and D.C. Schmidt: Techniques for Optimizing CORBA Middleware for Distributed Embedded Systems. *Proceedings of INFOCOM 1999*: 513-521.