3-2001

Infosphere Project: An Overview

Calton Pu
Georgia Institute of Technology - Main Campus

Jonathan Walpole
Oregon Graduate Institute of Science & Technology

Infosphere Project: An Overview

Calton Pu, Karsten Schwan
Georgia Institute of Technology
{calton,schwan}@cc.gatech.edu

Jonathan Walpole
Oregon Graduate Institute
{walpole@cse.ogi.edu}

Abstract

We describe the Infosphere project, which is building the systems software support for information-driven applications such as digital libraries and electronic commerce. The main technical contribution is the Infopipe abstraction to support information flow with quality of service. Using building blocks such as program specialization, software feedback, domain-specific languages, and personalized information filtering, the Infopipe software generates code and manage resources to provide the specified quality of service with support for composition and restructuring.

1 Introduction

1.1 Technology Push

Computer technology has advanced steadily in terms of CPU speed, memory and storage capacity. This reliable evolution of hardware, commonly known as Moore's Law, has made information technology the foundation of modern economy. This evolution is leading to a shared vision called ubiquitous computing [13] (see Section 5.3 for pointers to several ongoing research projects in this area). In a ubiquitous computing environment of the future, humans will be surrounded by hundreds to thousands of processors linked in a pervasive wireless network. At the same time, they will be surrounded by a huge amount of information. In the *Infosphere* project, we are building the systems support to bring the information closer to humans.

We first note the impact of technology push on information storage. Jim Gray mentioned in his Turing Lecture (Atlanta, May 1999) that the total disk storage capacity shipped last year was on the order of exabytes, which is about the same order of magnitude as the total of all previously available storage. Each 12 to 18 months, as the disk storage density doubles, vendors produce and have been selling new storage capacity equal to all of the previous storage combined. Gray also estimated that all professionally produced printed material in the entire human history would fit into a few exabytes. In the near future, we will be able to store many new kinds of information such as virtual reality, or more importantly in our view, massive amounts of up-to-date information about our physical reality in unprecedented detail. This new information comes from many sources including electronic commerce databases such as online banks, sensors from satellites, and smart appliances in ubiquitous computing.

1.2 Information-Driven Applications

Several important emerging classes of distributed applications are inherently information-driven. Instead of occasionally dispatching remote computations, such information-driven systems tend to transfer and process streams of information continuously. Member of this class range from applications that primarily transfer information over the wires such as digital libraries, teleconferencing and video on demand, to applications that require information-intensive processing and manipulation, such as distributed multimedia, Web search and cache engines. Other applications such as electronic commerce combine heavy-duty information processing (e.g., during the discovery and shopping phase, querying a large amount of data from a variety of data sources) with occasional remote computation (e.g., buying and updating credit card accounts as well as inventory databases).

In the Infosphere project, we are particularly interested in *fresh* information that changes the way we interact with our environment. For example, weather is considered an inevitably uncertain element of nature. Given fresh sensor information at suitably fine granularity (both in space and in time), accurate weather reporting and forecasting for small areas can be very valuable for everyone. For a birthday party in the park, an accurate weather forecast could mean happiness instead of disappointment. For farmers facing a sudden freeze, it may spell the difference between saving the crop or bankruptcy. For a military commander leading a rescue mission in a hostile country, it is a matter of life or death. Similarly, having an accurate picture of the highway conditions may mean arriving at work on time, beating the deadline for an important delivery, or reaching the hospital before the patient dies. While knowing the world in detail has been an expensive mission of agencies such as the CIA, CNN has shown us the value of up-to-date information about current affairs. We envision the fresh information providing more details

about the current state of our physical world than ever imaginable, impartially to all human beings.

The delivery of fresh information requires assured quality in the information flow. The success of guaranteed delivery services such as FedEx illustrates the importance of timely delivery in the physical world. Physical goods that are perishable usually have easily distinguishable criteria, such as sell-by dates, to separate the fresh from the decayed. Labeling goods in this way not only allows users to make informed decisions on whether or not to buy them, it also enables delivery mechanisms to make informed decisions about how to transport them. In general, many different properties are involved in such decisions. For example, the rate of decay for some produce is dependent not only on the amount of time it takes to reach the store, but also on transportation conditions such as temperature and humidity. Analogously, transportation conditions, such as constrained bandwidth, delay and noise, affect the concrete delivery properties of information flows which themselves span many dimensions including resource level concerns such as bandwidth, latency, and jitter as well as higher level concerns such as freshness, reliability, trustworthiness, security and survivability. Our challenge is to develop the systems that manage these concrete delivery properties for information flows all the way from information producers to information consumers.

Complex information systems depend on a number of fundamental components working together in a critical path. For example, the Internet infrastructure and hypertext were fully developed several years before the World Wide Web standards (HTTP and HTML) were approved. Similarly, the Web was deployed a couple of years before the Mosaic browser was released. However, extra credit is given to the Web browser, since it was the missing link in a long chain of breakthroughs that made the Web and the Internet finally take off. In the technology chain leading towards ubiquitous computing, we divide the concrete information flow into the producer end, the consumer end, and the missing link in the middle. On the information producer end, the network technology push and the resulting massive content generation (e.g., the Web) provide the supply of information. On the information consumer end, many dot.com and traditional companies are working on information appliances and application software that provide many different ways to access information. The missing link is the systems software that links networks to applications, including operating system, middleware, and data management layers.

1.3 Ubiquitous Information Vision

We call our vision *ubiquitous information*, in contrast to ubiquitous computing, since our focus is on the delivery of fresh information. Ubiquitous information goes beyond just gathering, storing, and retrieving increasing amounts of information. Living in a ubiquitous information environment means being in constant contact with both the physical world and the information civilization. For example, no one will ever get lost or become helpless, whether hiking in remote mountains or walking in the dark alleys of an inner city area. Help is always and immediately available through two-way low latency information flow to and from proper authorities, friends and family, or cyber-neighbors and cyber-Samaritans. Similar difficult situations arise from violent phenomena in our world such as flash floods, tidal waves, wild fires, volcanic eruptions, hurricanes, tornadoes, riots, wars, and terrorism. By providing detailed and up-to-date information about the actual situation, humans can handle these problems from a much higher vantage point. Even if we are still some years away from controlling these violent phenomena, the world will be a much more civilized place with ubiquitous information.

The Infosphere project is building the systems software support towards ubiquitous information, focusing on the proper treatment, transmission and delivery of information. While networking and communications researchers have been working on data transmission and delivery for many years, traditional computer science disciplines such as operating systems and programming languages have focused primarily on computation. By shifting our attention to proper information treatment and propagation, new and interesting research problems arise. Examples include delivery property management, adaptive resource management, and smart information delivery, as described in Section 2.

1.4 Client/Server Model

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. RPC is based on the procedure call abstraction which raises the level of abstraction for distributed systems programming beyond raw message passing and naturally supports a request-response style of interaction that is common in many applications. 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, DCOM, and CORBA. The client/server model is widely considered to be a good choice for building practical distributed applications, particularly those using computation or database backend servers.

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. Concretely, when information flows are subject to real-world timing constraints certain elements of distribution transparency – an

often-cited advantage of RPC – can cause more problems than they solve. For example, restrictions on the available bandwidth or latency over a network link between two components of a media-streaming application are a serious concern and should not be hidden by the programming abstraction. Similarly, the reliability and security-related characteristics of a connection may be significant to applications that are streaming critical or sensitive information. We refer to these characteristics as the *quality of service* (QoS) properties of an information flow.[1]

We argue that an appropriate programming paradigm for information-driven applications should embrace information flow as a core abstraction and 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 increasing size, complexity and heterogeneity of information-driven applications. We emphasize that such a new abstraction offers an alternative that complements RPC, *not* to replace it. In client/server applications, RPC is clearly the natural solution.

1.5 The Infosphere Approach

We propose the *Infopipe* abstraction to link information producers to information consumers. In addition to their basic function of transporting information, Infopipes manage and manipulate the concrete delivery properties of the information flowing through them, such as freshness. Infopipe creation and composition involve the specification of the syntax, semantics, and QoS properties. Infopipe execution requires system resource management mechanisms to maintain the QoS properties. The Infopipe specifications are translated automatically by the system into an actual implementation with the desired behavior. When information flows through an Infopipe, concrete delivery properties and requirements such as freshness, performance, and security are updated en route and maintained each step of the way.

Specifically, application designers will specify QoS property requirements by explicitly describing the requirements of an information flow. These explicitly defined properties are used by the system to generate code that maintains QoS during delivery. In this way QoS requirements define policies appropriate to the type and use of information flow. Because of real-world constraints over information flow speed, time, capacity and cost, QoS property management is generally concerned with trading among properties to preserve the most important properties at the expense of the others. This trading approach requires adaptive resource management algorithms.

Infopipes fill the missing link to form the foundation of a new paradigm for building information-driven applications. We call the concrete software architecture that implements the information flows the *producer/consumer* architecture, in contrast to the traditional client/server architecture. The main difference between the two is the emphasis on information flow in the Infopipes of the producer/consumer architecture, as opposed to the control flow focus of RPCs in the client/server architecture. By focusing on information flow in this way we are able to manage delivery properties explicitly. In the producer/consumer architecture, information generated by a producer is carried to consumers by Infopipes. Thus ubiquitous information environments will be built from a network of dynamically adapting Infopipes linking information producers and consumers.

2 Systems Support for Information Flow

2.1 Infopipe Concept

We propose the *Infopipe* as an abstraction for capturing and reasoning about information flow in information-driven applications. Intuitively, an Infopipe is the information dual of an RPC. Like RPCs, Infopipes raise the level of abstraction for distributed systems programming and offer certain kinds of distribution transparency. Beyond RPCs, Infopipes have attached QoS properties that allow control over the quality, consistency, reliability, security and timeliness of the information flowing through them. Furthermore, the Infopipe concept has inherent data parallelism and is concerned with a high level abstraction that embraces content semantics and user requirements in order to control information flows and optimize resource consumption. This distinction becomes particularly significant when considering QoS properties such as the quality or consistency of a flow of information.

A simple Infopipe has two ends – a consumer (input) end and a producer (output) end – and implements a unidirectional information flow from a single producer to a single consumer. The processing, buffering, and filtering of information happen in the middle of the Infopipe, between the two ends. As mentioned before, an Infopipe links information producers to consumers. The information producer exports an explicitly defined information flow, which goes to the input end of the Infopipe. After appropriate transportation, storage, and processing, the information flows through the output end to the information consumer.

[1] We use the term quality of service and QoS in a broad sense, including many systemic properties such as performance, availability, and security. This includes the initial definition of guaranteed QoS through reserved resources (e.g., in network bandwidth) as a special case.

Infopipe is a language and system independent mechanism to process information in a distributed system. This is done on purpose since one of the main reasons for RPC's success among practical imperative programming languages is their universal adoption of the procedure call abstraction. As a consequence, stub generators are able to hide the technical difficulties of marshalling and unmarshalling parameters for all practical languages. There are two additional sources of problems in the implementation of stub generators: (1) the heterogeneity of operating systems and hardware, and (2) the translation between the language level procedure call abstraction and the underlying system level message-based implementation. The eventual definition of an Interface Description Language (IDL) solved both problems, by encapsulating the translation functions in a portable IDL compiler.

Our approach to making Infopipes language and system independent parallels that used in RPC. We define a generic interface for Infopipe manipulation, and use the equivalent of IDL and stub generators to hide the technical difficulties of marshalling and unmarshalling data and manipulating system-specific mechanisms for QoS property enforcement. By adopting this approach we shield the application developer from the complexity of heterogeneous operating systems and hardware and the translation from language-level abstractions to underlying message-based implementations.

In the same way that RPCs form the foundation of the message-oriented client/server architecture, we envision Infopipes to be the basic building blocks for information-driven distributed applications. In contrast to the client/server architecture, in which message-based transactions among clients and servers are the primary concern, we call the distributed information-intensive processing architecture the producer/consumer architecture, since we are primarily concerned with the information flows from producers to consumers.

2.2 Infopipe Typespec

Typespec (a refinement of the type concept in programming languages) defines an Infopipe's consumer and producer ends. A Typespec consists of the explicit description of the syntax, semantics, and QoS requirements of the information flow. The syntactic part is equivalent to the schema of information flow. We divide the information flow into successive *units* and use the term schema in the database sense, i.e., a detailed description of the unit of information. Users interpret the information flow using the Typespec schema and the semantic description. The third component of Typespec consists of the QoS requirements, for example, bandwidth, latency, and jitter for multimedia applications. Typespec defines which properties are meaningful for an Infopipe. A QoS property defined in the Typespec usually has three parts: (1) a specification of what the property should be, (2) predefined limits of that property, if any, and (3) the actual, current reading of that property. For example, an Infopipe may have a specified bandwidth of 1Mbyte per second (specification), on a network with T1 (1.5Mbyte/sec) maximum bandwidth, and data currently flowing through it at 1.1Mbyte per second (actual reading).

Between its consumer and producer ends, an Infopipe is a one-way mapping that transforms information units from its input domain (consumer Typespec) to the output range (producer Typespec). Probably it is not surprising to the reader that there are many examples of concrete Infopipes in existing systems. We are particularly interested in generic Infopipes that are programmable using a domain specific language, since this is our design choice for Infopipes in general. The first group of examples is in Unix filter programs. The `sort` program is a permutation mapping from one `stdin` record to one `stdout` record, and `sort` offers a domain specific way (invocation parameters) for specifying the permutation transformation. Another Unix filter example is subset selection mapping, for instance, done by `grep` or `egrep` that use regular expressions for the subset specification. These are examples of mappings in which their domain is the same as range.

There are several concrete examples of mappings between a domain that is different from the range. Again using Unix filters, the line editor `sed` is a good example of transformation from one format into a different format under an editing script control. In general, Unix pipeline is a clear example of the Infopipe programming style. A technical difference is that Unix streams are byte-oriented (untyped), while we advocate Typespec conformance for Infopipes (see below). Another important concrete Infopipe example is represented by relational databases. Consider SQL as a domain specific language that defines all access operations to a relational database Infopipe with an active interface. Instead of invoking a read operation, we send an SQL program to the database, and it returns the information requested. The database transforms the database schema into the result format specified by the SQL query, performing selection, projection, and join operations as part of the transformation.

Infopipe run-time system implements the Typespec description through a careful management of system resources. We motivate the flow properties in Typespec with an analogy. Consider oil flowing in a pipeline at a constant speed. The cross section of pipe multiplied by the flow speed is analogous to the bandwidth of information flow (e.g., in a network). The length of the pipe divided by the flow speed also gives us the latency between entering into one end of the pipe to exiting from the other. The flow speed can be calculated by considering the viscosity of oil, the friction on internal pipe surface, and pressure caused by gravity or a pump. Merging two pipelines into one (or splitting one into two) changes

these variables and flow behavior in predictable ways. For example, merging two pipes requires a resulting pipe with their combined cross sections to maintain the same flow speed. Similarly, combining or dividing an Infopipe may cause changes in bandwidth or latency. One of our research goals is to preserve important flow properties such as bandwidth, latency, and (lack of) jitter.

We say that the consumer end of Infopipe A's Typespec *conforms* to the producer's end of Infopipe B's Typespec when the following conditions hold:

1. The definition of operations of Typespec A is a subset of the operation's definition of Typespec B.
2. The schema of Typespec A is compatible with the schema of Typespec B, i.e., a unit of B data can be interpreted using the schema of Typespec A.
3. The specification part of Typespec A's properties falls within the range of the limitations part of Typespec B's properties.

The first condition says that any operations that A might invoke (or a domain specific microprogram sent) will be understood and supported by B. The second condition says that a unit of B data can be interpreted by Infopipe A, although A may or may not interpret B data at the fullest level of detail. The third condition says that B has a theoretical chance of satisfying A's requirements for all flow properties of interest to A.

The Typespec of Infopipe A's consumer end must conform to the Typespec of Infopipe B's producer end for them to be connected. The three conformance conditions do not guarantee that the resulting Infopipe pipeline will always function perfectly, but they provide the initial conditions to get the Infopipe pipeline started flowing.

Typespec is the cornerstone for Infopipe reuse. From the system construction point of view, one of the fundamental difficulties in software reuse is in the myriad of small differences in the details. Despite encapsulation (e.g., in object-oriented systems), typical software modules make many assumptions that may or may not fit the new application, where it is being reused. The situation is aggravated by composition when these assumptions become buried under many layers. We use Typespec to reveal the assumptions made by each Infopipe and to describe its functional capabilities as well as limitations.

2.3 Infopipe Composition

One of the most important requirements in large-scale distributed applications is their scalability to ever increasing number of nodes and users, as well as continual addition of new functionality into the software. Composition of software modules is one of the established methods to add new functionality, assuming that those modules are compatible and cooperative with each other. Unfortunately, the composition of software modules based on procedural interfaces (e.g., client/server systems using RPC) frequently becomes limited by the complexity of interfaces involved. Given the relatively low level of abstraction of those interfaces, this limitation is understandable and probably unavoidable. Note that the limitation is due to the complexity itself, and interface standards such as CORBA do not address this problem.

In contrast, Infopipe is designed to be composable, i.e., the building of an information flow pipeline by connecting component Infopipes together. By composable we mean two things. First, the consumer of one Infopipe should connect easily (through standard and simple interfaces and meta-interfaces) to producer(s) of other Infopipe(s). This is captured by Typespec conformance as defined in Section 2.2. Second, the Typespec of the composite Infopipe should be derivable from the Typespec of the component Infopipes. Consequently, those Typespec components that are easy to derive under composition are "better" in some sense compared to Typespec components that change in unpredictable ways when composed.

Bandwidth and latency are examples of good Typespec component specifications under composition. In general, the composite Infopipe will have the smallest bandwidth of all the segments from an Infopipe pipeline. Analogously, the latency of a composite Infopipe is calculated by the addition of the latencies of the component Infopipes. Jitter is an example of Typespec that has more complex behavior than latency. Adding deep buffers into an Infopipe, for example, may reduce jitter but it increases end-to-end latency. If we reduce the total amount of buffering, the latency is reduced but jitter may be compounded by ripple effects adding up small hiccups in each component Infopipe, when data availability is not smooth. A possible calculation of the composite Infopipe jitter (assuming sufficient other resources such as CPU) is a cumulative addition of jitter functions, where the depth of each stage (buffer size) is subtracted from the partial sum up to that stage. Real-time constraints, e.g., fixed arrival time often restricts the number/kind of intermediate stages in the composite Infopipe.

2.4 Infopipe Restructuring

Some of Typespec properties concern the internal structure of Infopipes. For example, an Infopipe may have some internal data buffering. This is common for many reasons, including performance and jitter reduction. While buffering may reduce jitter, the depth of the buffer may introduce additional latency in data transmission. Therefore, even though the depth of Infopipe may appear to be an implementation detail from some angle, it affects other important Typespec properties and so we include this kind of properties as an integral part of our Typespec research. Another example of Typespec property that is related to Infopipe structures is the abstract semantics of information unit ordering. Up to now we have made the common assumption that an information flow is FIFO

(first in, first out). In an Infopipe, however, this assumption can be relaxed to include other common semantics from data structures such as queues (e.g., unordered queues), stacks (e.g., LIFO), or network properties (e.g., reliable delivery).

For simplicity of presentation, we have made an implicit assumption that Infopipes have one producer end and one consumer end. This is what we call *1-1* Infopipe, in terms of external structure (interface). We now introduce a natural classification of Infopipe structure into four groups.

- **1-1 Infopipe** (*single producer and single consumer*). Since most of the practical Infopipes are of 1-1 variety, we omit the numbering when referring to 1-1 Infopipe.
- **N-1 Infopipe** (*multiple producers and single consumer*). This Infopipe receives data from several Infopipes, combines them, and transmits all to a single consumer.
- **1-M Infopipe** (*single producer and multiple consumers*). This Infopipe takes the information units from one producer and transmits them to M consumers. Sometimes the information is replicated (e.g., multicast) and other times the consumers see different data (e.g., multiple consumers of the same queue) from the same source.
- **N-M Infopipe** (*multiple producers and multiple consumers*). While we could make this case a separate group, for simplicity we reduce it to a composition of an N-1 Infopipe with a 1-M Infopipe.

This classification covers Infopipes with a known number of producers and consumers. Some Infopipes are *open*, i.e., they have an unknown number of producers or consumers. Concretely, typical broadcast media fall into this category. A producer transmitting over a satellite link, for example, can be received by a large number of consumers. We model this situation as a 1-M Infopipe with implicit message replication.

Composite Infopipes may be relatively simple, for example, a pipeline of 1-1 Infopipes forming a 1-1 composite. Conversely, another composite Infopipe may have several N-1 and 1-M components that result in a complicated information flow. In Section 2.3, we described the rules of Infopipe composition and Typespec conformance, so the properties in the composite Infopipe Typespec can be calculated. Once constructed, a composite Infopipe can be dynamically restructured using the same Typespec conformance rules. In analogy to previous work on activity restructuring [2], the main restructuring meta-operations are split, join, link, and modify:

- **Split** – divides an Infopipe producer or consumer end into two identical ones.
- **Join** – merges two similar producer or consumer ends into one.
- **Link** – connects the producer end of one Infopipe with the consumer end of another Infopipe.
- **Modify** – changes the Typespec using application semantics or system information.

3 Infopipe Implementation

3.1 Implementation Strategy

The first part of the Infopipe implementation is the Infopipe correspondent of stub generators for RPC. The main difference between RPC and Infopipe is the procedure call's single level of abstraction in RPC compared to multiple abstractions in Infopipe, for example, introduced by Infopipe structures and QoS specifications. To address this issue, we are applying the concepts from Aspect-Oriented Programming (AOP) developed by Kiczales' team at Xerox PARC, to isolate the various aspects of Infopipe abstraction. These abstraction aspects will be captured and implemented by the Infopipe Specification microlanguage (ISL) using domain specific language technology. Since Typespec contains the concrete description of these abstraction aspects, the function of ISL is to support a compiler that transforms the Typespec information into plumbing code similar to marshalling and unmarshalling code in stub generators.

In Section 2.2, Typespec was divided into three components: syntax, semantics, and QoS properties. The software that wraps the first two components corresponds directly to RPC stub generators, since the ISL will generate the plumbing code so Infopipe programmers don't have to write code to manipulate the explicit representation and description of an Infopipe. In addition, the QoS properties such as security and performance may be subdivided into aspects and their handling wrapped by ISL compiler. Infopipe programmers need only to specify, for example, a range of acceptable bandwidth, and the ISL will generate code to maintain that bandwidth and notify QoS exception handling mechanisms when the actual bandwidth falls outside the specified range.

We have built a prototype and several components of ISL software, in particular the ISL Stub Generator for information flows in the XML format. We are in the process of building more robust versions of ISL software for experimentation by our collaborators as well as the community at large. The next step is the research on Infopipe composition.

We separate out the composition of Infopipes into an Infopipe Composition and Restructuring Microlanguage (ICRL). In Sections 2.3 and 2.4, we outlined the composition and restructuring operations for Infopipes. These operations will be defined in an ICRL so Infopipe programmers can build and modify Infopipes easily. ICRL microprograms are then translated into ISL and Infopipe

linking/composition code. We plan to implement a family of ICRLs, each with increasing sophistication, to allow rapid development and testing of Infopipe concepts. For example, the first member of the ICRL family is likely to support only the composition of simple (e.g., 1-1) Infopipes.

3.2 Technology Building Blocks

Given our emphasis on in the modularity and composition of Infopipes, a natural question from the systems community is performance. If we were to build Infopipes the traditional way, modularity would have implied high composition and layering overhead. The first building block we use is the specialization technology to reduce execution overhead through systematic program transformations developed in the Synthetix project [7, 9], among others. A relevant example is the automated specialization of SUN RPC code using the Tempo-C specializer [8]. This is a demonstration of specialization technology on production software, by cutting out layering overhead using program analysis automated in Tempo-C.

In addition to C and Java, we have used Tempo to implement very efficient specialized interpreters for domain specific languages, which form the second building block of Infopipe construction. Two examples are the GAL microlanguage for writing graphics device drivers and the PLAN-P microlanguage for the construction of active network protocols. These specialized interpreters have performance comparable to production just-in-time compilers (e.g., Java), but are much easier to write and specialize. Our implementation plans are designed to make use of these proven specialization techniques and tools to improve Infopipe performance while preserving microlanguage-level modularity.

An important requirement for a software system to run smoothly in a shared environment, such as the Internet, is the ability to adapt to both sudden and gradual changes in environmental variables such as network congestion and noise. The resource management layers of Infopipes will use feedback-based schedulers, the third building block, to adaptively manage shared resources. Where necessary, appropriate property management policies are invoked to deal with the constraints of over-loaded resources. We will use our software feedback toolkit to construct basic feedback components with well-defined behavior. During Infopipe composition, these basic components will be composed into more sophisticated feedback mechanisms with predictable behavior and failure modes. Infopipes equipped with feedback are able to adapt to significant environmental changes. For example, simple feedback-based schedulers can adjust the resources given to the processes in a pipeline to keep the information flowing smoothly, despite fluctuations in the flow bandwidth and arrival times.

In addition to fine-grain adaptation, feedback toolkit-based implementations have two other advantages. First, the adaptive capability of component Infopipes is used in predicting the adaptation capability of composite Infopipe. Recall that Typespec describes the functional capabilities and limitations of each Infopipe, component and composite alike. In particular, feedback-based adaptation has well-defined limitations in their applicability, which are explicitly documented in the Typespec. The second advantage is the ability to detect and recover from feedback failures. We use *guards* to detect these situations and *replug* failing feedbacks with more generic control and adaptation mechanisms (e.g., perhaps more stable although less responsive control policies such as admission and congestion control through job cancellations). Guarding and replugging are techniques well understood in our research on specialization and feedback [11, 12].

In the next section we outline some concrete applications that will benefit from Infopipe architecture and implementation. These applications are not only scenarios that motivate our research. We have active projects that are building software for these applications. For example, the update monitoring application (Section 4.1) is realized by the Continual Queries project [6, 5]. The distributed multimedia example (Section 4.2) is realized by the Quasar project. The successive versions of Infopipe implementation will be evaluated in concrete applications that we have been working on.

4 Application Scenarios

4.1 Update Monitoring

The deluge of Internet information shows the human limitations of browsing in finding the right information. Consequently, the discovery of fresh information is best achieved through automated monitoring. In traditional database applications, update monitoring has not included freshness or other delivery properties, since these properties are “outside the database”. The only guarantee that traditional update monitoring systems provide is to deliver information updates when they reach some specified conditions of interest. In contrast, fresh information delivery in the ubiquitous information requires two levels of filtering: the conditions of interest and the freshness requirement. Conditions of interest then trigger active push information delivery mechanisms. Furthermore, fresh information delivery needs to keep track of not only the arrival of fresh information (conditions of interest) but also the decay rate of fresh information.

Our experience with the research and development of the Continual Query (CQ) system [6] demonstrates that update monitoring at Internet scale is best implemented as Infopipe systems from both user perspective and system extensibility and QoS control perspective. For example,

we can model each user's update monitoring request as an Infopipe with the user as the consumer end and the CQ server as the producer end. Once the Infopipe is installed and activated, the CQ system will invoke the query routing service to find the set of relevant data sources that can handle the given request and then invoke the query parallelization manager to generate an optimized parallel query execution plan. Such a parallel query plan is produced by dynamically splitting the user's request Infopipe into a collection of data source-specific Infopipes, each targeted at one relevant data source, and has the CQ server as the consumer end and the specific data source as the producer end. The quality of service requirements of each Infopipe is entered using the Typespec and automatically guaranteed by the Infopipe QoS manager.

Application specific update monitoring is already in widespread use in many practical applications. For example, electronic trading companies such as ETRADE send stock market updates, and airlines send weekly special ticket sale prices. Our goal is to provide this capability in a generic way to many information-driven applications, for example, in digital libraries and electronic commerce.

4.2 Distributed Multimedia

Multimedia systems are fundamentally information-flow systems with demanding QoS requirements. They often exhibit complex streaming structures among multiple clients and servers, for example, involving dynamically reconfigurable broadcast and multicast distribution trees. These structures are difficult to construct using the pairwise request-response constructs of RPC-based models. Furthermore, multimedia systems often require direct manipulation of the data within a stream, for example to decode it, or transcode it to adapt its quality, and they often require real-time synchronization among and within streams. For these reasons streams cannot be treated as completely opaque. Hence the levels of transparency associated with object binding in RPC-based systems tend to hide too many details of the underlying environment that affect these properties. QoS management involves the specification of QoS requirements and tolerances, negotiation of QoS contracts, run-time actual QoS monitoring, policing and dynamic renegotiations, and specification of QoS trade-offs and dynamic adaptation policies. Infopipes offer great leverage in all of the areas mentioned above by defining interfaces that export control to the application in high-level, domain-specific terms, and by automatically generating "stubs" to hide the complexity of managing the details of the interaction with lower layers.

Videoconference systems require a variety of multimedia stream properties that can be represented as QoS properties in Infopipe Typespec. Examples include frame-rate, frame jitter, end-to-end latency, frame/data drop ratios and synchronization among audio and video streams. Despite many attempts, current videoconferencing software packages are still limited in performance and quality when running over shared environments such as the Internet. For example, frame-rate is a domain-specific notion of the temporal resolution of the video stream and can be used together with other information (such as spatial resolution and encoding format details) to calculate the bandwidth requirements of the stream. Infopipe can translate frame-rate requirements to bandwidth requirements, hence hiding complexity from the application developer. End-to-end latency is clearly visible in videoconference systems. Infopipe stubs will calculate the accumulated latencies from various system components, compare them to the overall end-to-end latency requirements, and make the necessary adjustments.

We will demonstrate and experiment with ubiquitous information by integrating information producers and consumers. The integration starts with the wrapping of fresh information sources on the Internet and filtering interesting fresh data to form information producers. Data sources such as sensors or e-commerce databases produce a continuous stream of fresh data and therefore are of particular interest. On the consumer side, we will build the software for lightweight and inexpensive combinations of communications devices (e.g., a cell phone), computers (e.g., a personal digital assistant), and sensors (e.g., a GPS receiver), which will serve as unobtrusive connections between users and the ubiquitous information. The Infopipe backbone will connect the producers to the consumers with well-defined delivery properties.

4.3 Current Status

The main results at Oregon Graduate Institute have been primarily in the Kernel Layer, divided into two areas. The first area is the research on Real-Rate Infopipes. We have defined a "QoS-adaptive real-rate network service" which will form the basis of real-rate infopipes. Concretely, we have been working on the definition of packet formats for real-rate flows, understanding how traffic specifications should work, and more importantly, understanding how to parameterize feedback control mechanisms and provision buffers so that real-rate scheduling can work along a pipeline. Some of research challenges have been summarized in a paper [10] using environmental observation and forecasting as a concrete application.

The second area is the study of formal system properties when feedback is used. For example, we have been developing a feedback-based model of TCP-friendly congestion control – basically, using feedback control ideas to understand the fine-grain dynamic behavior of TCP-like congestion control so that we can produce other real-rate transmission protocols that are truly TCP-friendly. In addition to simulation studies, we have modified the Linux kernel to let users adjust any TCP flow's AIMD in-

crement and decrement parameters, allowing them to produce TCP flows with various levels of aggressiveness.

The main results at Georgia Institute of Technology have been made primarily in the Middleware and Application Layers. The Middleware layer work is both conceptual and practical. Conceptually, we have been describing and decomposing information-driven applications. Practically, we have been building efficient Infopipe support based on the Echo and JEcho, a publish/subscribe communications facility developed previously.

At the Application Layer, we have been developing and adapting tools to support personalized filtering of information [3]. We also have been developing an Infopipe Stub Generator (ISG) using XML as the transmission format. The ISG generates code to interpret and create XML data streams for applications that transform XML data. The ISG is now being developed for BPIO, a binary format used by Echo and JEcho.

Another area of research at this layer is the methods and software to extract information automatically from the Web [1, 4]. This is an important preparatory step for most of data, since we need to add the Typespec description to the data stream before it can be transported by Infopipes. The automation of the information extraction process (Usually in the form of wrapper generators) is crucial to the gradual importation of non-Infopipe data, since all information originated from the real world (e.g., sensors) probably will require some kind of wrapping and cleansing.

The work on the middleware level Infopipe consists of three parts. First, the definition of the middleware Infopipe model, interfaces, and a specification language for the definition of middleware Infopipes. Second, the implementation of a prototype middleware Infopipe for evaluation and validation. Third, the interactions between the middleware Infopipe with the other levels (e.g., the kernel level Infopipe below and the application level Infopipe above) in terms of interfaces and integration. The current priorities are with the first and second parts.

The work on the application level Infopipe consists of three parts. First, the definition of the application Infopipe model, interfaces, and a specification language for the definition of application Infopipes. Second, the implementation of a prototype application Infopipe for evaluation and validation. Third, the interactions between the application Infopipe with the lower levels (e.g., kernel level and middleware level) in terms of interfaces and integration. The current priorities are with the first and second parts

We are currently redesigning the Infopipe software to retarget it to embedded devices such as PDAs. This includes the exploratory adoption of AspectJ compiler and AOP methodology, the development of translation routines between XML and BPIO for middleware implementation, and development or adoption of embedded applications for evaluation. We are also investigating the use of program specializers such as Tempo-C and Tempo-J for optimizing embedded code while preserving portability and maintainability.

5 Appendices

5.1 Funding Acknowledgements

The Infosphere project is funded by DARPA/ITO through the Information Technology Expeditions, Ubiquitous Computing, Quorum, and PCES programs. The PI and co-PIs are also partially funded by NSF's CISE directorate, through the ANIR and CCR divisions. In addition, the PI and co-PIs are partially funded by Intel and Tektronix.

5.2 Project Scope

A project of the scale of Infosphere necessarily involves many participants organized as tasks. In the Base/QoS tasks, the main faculty members are: Calton Pu (principal investigator), Ling Liu, and Karsten Schwan of Georgia Institute of Technology, and Jonathan Walpole of Oregon Graduate Institute. In the Context Aware task, the main faculty member is Gregory Abowd of Georgia Tech. In the Embedded Infopipe subproject, two additional faculty members participate: Mustaque Ahamad and Yannis Smaragdakis of Georgia Tech. Other research staff members include Greg Eisenhauer, a research scientist, and J. Adam Sigler, a research programmer at Georgia Tech.

Many graduate students have been working in the many aspects of the project. They include: Fabian Bustamante, David Buttler, Wei Han, Henrique Paques, Galen Swint, Wei Tang, and Patrick Widener of Georgia Tech, and Ashvin Goel and Kang Li of Oregon Graduate Institute. In addition, several visiting scholars have contributed to the project, including Prof. Yasushi Shinjo of University of Tsukuba (Japan), and Mr. Koichi Moriyama of Sony Corporation. Mr. Moriyama built the first prototype of the Infopipe Stub Generator during an extended visit to Georgia Tech.

We also have several collaborators working with us on research topics related to Infosphere technology and goals. They include Prof. Molly Shor of Oregon State University, Prof. Charles Consel and Prof. Gilles Muller of University of Bordeaux (France) and Prof. Masaru Kitsuregawa of University of Tokyo (Japan).

5.3 Related Project Pointers

There are four other Information Technology Expedition projects that are exploring the long-term impact of technology that leads to the vision of ubiquitous computing

and ubiquitous information. Pointers to the expedition projects are:

- Aura Project http://www.cs.cmu.edu/~aura/ at Carnegie-Mellon University (PI: Prof. David Garlan http://www.cs.cmu.edu/~garlan).
- Endeavour Project http://endeavour.cs.berkeley.edu at University of California at Berkeley (PI: Prof. Randy Katz http://www.cs.berkeley.edu/~randy).
- Infosphere Project http://www.cc.gatech.edu/projects/infosphere at Georgia Institute of Technology and Oregon Graduate Institute (PI: Prof. Calton Pu http://www.cc.gatech.edu/~calton).
- Oxygen Project at Massachusetts Institute of Technology (PI: Prof. Michael Dertouzos http://www.lcs.mit.edu/about/mld), with several presentations available at http://www.lcs.mit.edu/anniv/.
- Portolano Project http://portolano.cs.washington.edu/ at University of Washington (PI: Prof. Gaetano Borriello http://www.cs.washington.edu/homes/gaetano/).

Related project URLs:

- Quasar project (Section 4.2) URL: http://www.cse.ogi.edu/DISC/projects/quasar.
- Continual Queries (Section 4.1) URL: http://www.cc.gatech.edu/projects/disl/CQ/.

6 References

Space constraints prevent us from listing all the relevant references in the literature and on the web. We include some papers for illustrative purposes.

1. David Buttler, Ling Liu, and Calton Pu, ``A Fully Automated Object Extraction System for the World Wide Web", to appear in the *Proceedings of the 2001 International Conference on Distributed Computing Systems (ICDCS'01)*, May 2001, Phoenix, Arizona.
2. L. Liu and C. Pu, "Methodical Restructuring of Complex Workflow Activities", *Proceedings of the IEEE 14th International Conference on Data Engineering (ICDE'98)*, Orlando, Florida, February 1998.
3. 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.
4. L. Liu, C. Pu, and W. Han, "An XML-Enabled Data Extraction Tool for Web Sources". To appear in the special issue on Data Extraction, Cleaning, and Reconciliation of *Information Systems: An International Journal*, Kluwer Academic, 2001.
5. Ling Liu, Calton Pu, Wei Tang, and Wei 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.
6. Ling Liu, Calton 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.
7. Dylan McNamee, Jonathan Walpole, Calton Pu, Crispin Cowan, Charles Krasic, Ashvin Goel, and Perry Wagle, Charles Consel, Gilles Muller, and Renaud Marlet, "Specialization Tools and Techniques for Systematic Optimization of System Software", *ACM Transactions on Computer Systems*. To appear in the May 2001 issue.
8. 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.
9. Calton Pu, Tito Autrey, Andrew Black, Charles Consel, Crispin Cowan, Jon Inouye, Lakshmi Kethana, Jonathan Walpole and Ke Zhang, "Optimistic Incremental Specialization: Streamlining a Commercial Operating System", *Proceedings of the Fifteenth Symposium on Operating Systems Principles (SOSP'95)*, Colorado, December 1995.
10. David Steere, Antonio Baptista, Dylan McNamee, Calton Pu, and Jonathan 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.
11. D. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-Driven Proportion Allocator for Real-Rate Scheduling", *Proceedings of the Third Symposium on Operating System Design and Implementation (OSDI'99)*, New Orleans, February 1999.
12. David Steere, Molly H. Shor, Ashvin Goel, Jonathan Walpole, and Calton Pu, "Control and modeling issues in computer operating systems: resource management for real-rate computer applications". *Proceedings of the 39th IEEE Conference on Decision and Control (CDC)*, Sydney, Australia, December 2000.
13. Mark Weiser, "Some Computer Science Problems in Ubiquitous Computing," *Communications of the ACM*, July 1993.