1. Introduction

The purpose of this project is to build an e-commerce application using the Infopipes system and methodology. The Infopipes system is one abstraction of the Infosphere project, a joint effort between the Georgia Tech College of Computing and the Computer Science and Engineering Department at Oregon Graduate Institute [1]. Infosphere is interested in the quality delivery of data over distributed systems. In this project we attempt an implementation of an e-commerce application using the Infopipes architecture.

In the next section of this paper I describe the Infosphere project in general and explain the basic goals and architecture of the Infopipes abstraction. Section three describes the details of the e-commerce implementation as well as the specific contributions to the Infopipes system. The fourth section lists some possible next steps for completing the system.

2. Infosphere and Infopipes

The Infosphere project is all about the “Smart Delivery of Fresh Information” [1]. The Infosphere project is based on several key observations:

- Moore’s law still holds true meaning computers are more powerful and cheaper than ever before,
- Computers are present everywhere, including embedded systems such as PDAs, cell phones, and sensors,
- Networks have become extremely fast and very large areas may now be linked by fast Internet connections, and
- The amount of information available to humans has grown significantly [2].

The combination of these observations opens the door to a number of research questions and requires computer scientists to adopt a new perspective for conducting research. In particular, due to the large number of disparate and heterogeneous data sources available, the computational models of traditional computer science are no longer valid. How can we make sense out of all the available data? The goal of Infosphere is to tackle this question from a information flow perspective [2].

The primary abstraction of this data-centric paradigm is an Infopipe. Infopipes are compose-able pieces of middleware that accept data, process it, and then return it (possibly in another form). Emphasis is placed on developing local algorithms for each
pipe, and then composing pipes into larger, more complex systems. The Infopipes architecture is flow-oriented, handles rapidly changing and heterogeneous data, and dynamically adapts to changing conditions [2].

When designing a system using Infopipes, the programmer uses an XML language called XIP to specify all the pipes that will make up the system. Specifically, a XIP file contains the following information:

- The data types that will flow throughout the system,
- A description of the individual Infopipes,
- The inports and outports of each pipe,
- How pipes are composed to form higher level components, and
- The communication method used among the pipes.

A completed XIP file is then run through the Infopipe Stub Generator (ISG) to generate stubs and communication code for each of the pipes in the system. The ISG is designed to generate pipes in any programming language, although C and C++ are the only languages currently supported. After the ISG has completed, the developer takes the generated stubs and adds in the application’s core functionality. The system is then compiled using a generated Makefile and executed on either a single machine or distributed across many machines.

A key component of the Infopipes architecture is its support for Aspect Oriented Programming (AOP) [3]. The AOP model recognizes that there are certain “aspects” in a program that cut across several components or layers of the system. For example, consider an object-oriented program written in Java. Each of the objects represents an encapsulated component of the system. However, if we want to add logging to the system we would need to modify many of the classes individually and add in the logging code. Since the code added to each of the files will have a similar pattern, it would be helpful to automatically incorporate the logging “aspect” across the entire program. The Infopipes architecture supports the Aspect Oriented Programming paradigm by allowing the user to specify points in the XIP file where certain aspects should be “woven” in. When the ISG generates the system stubs, the aspect code is automatically inserted into the correct places [3]. This encourages reusability because after an aspect has been written for one system, it can then be reapplied in other systems.

Before beginning this project, most of the programs built using Infopipes were media-based applications. For example, imagine a streaming video source attempting to deliver video data to a PDA. Because the PDA has limited bandwidth and a small screen size, the data must be manipulated and changed into a form usable by the PDA. The data is passed through several Infopipes that throttle throughput and reduce the quality of video data so it can be effectively displayed on the device.

### 3. E-Commerce Application
Our goal in this project is to develop an e-commerce application using the Infopipes design methodology and toolkit. Due to the large size of the system, this project is a joint effort between two 7001 students, Yong Yang and me. The application has the following components:

- A login screen that allows users to enter their credentials,
- A database back end for retrieving and storing user information,
- A smart buffer that will hold a user’s data after he or she logs in, and
- A display for returning data back to the user.

Additionally, in order to reduce system load, we designed a “prefetch” operation to automatically retrieve important information from the database after the user logs in and store it in the smart buffer. When possible, requests for user information would be handled by the smart buffer instead of the database, therefore reducing the overall amount of traffic going to the database.

After receiving the project description, our first step was to determine which Infopipes we would need to build the system, and how we would divide the work between the two of us. Figure 1 shows the entire system and which team member is assigned to each Infopipe. I concentrated on the database related components, and Yong focused on the Smart Buffer related components.

![Figure 1](image-url)

After completing the initial design, we determined which data types we would need to pass between the various Infopipes. We decided on the following data types.

- **LoginInfo** – contains a username and password
- **LoginResult** – contains the username and a result field indicating whether or not the login was valid
The data types were then put into a XIP file and input into the ISG to verify they were valid. After completing the data types we could divide up the work between the two of us as indicated in Figure 1.

The next step was to create XIP files for each of our Infopipes. Because of the complexity of the system, we needed special pipes to provide plumbing and direct the data flow among the main components of the system. The DB Multiplexer pipe has two inports and one outport. This was needed because data requests were coming from two separate components and needed to be integrated into one request for the database. Although the Infopipes architecture was designed to support such a pipe, this was the first time a multiplexer was used in an actual implementation. When running this pipe through the ISG we discovered a bug preventing the generated code from accepting two inports. We discovered that the same problem would also exist for pipes with multiple outports. After fixing the bug, we were able to generate stubs that correctly handled multiple inports and outports.

Another type of pipe that had previously not been implemented in the system is the DuplicateData pipe. This Infopipe is essentially the opposite of a multiplexer: it accepts one input and forwards the data along two outports. In our design, we hard coded the data type of the inport and outports; however, ideally we would like the DuplicateData pipe to dynamically adapt to the input type and automatically output data of the same type. This is a significant challenge because the XIP files require an explicit data type for each port. Adding support for dynamic types would require changing both the XIP language specification, and the transformation stylesheets that generate the stubs.

After all XIP files were specified completely we needed to compose the individual pipes into a meaningful system using a master XIP file. We decided on a flat implementation in which all the components are put together on the same level. The system also supports hierarchical composition of pipes in which atomic pipes are put together to form small components, small components are combined to form larger components, etc., until the entire system has been constructed. The hierarchical approach requires more XIP code, but offers the advantages of having several layers of abstraction, and allowing developers to think of groups of pipes as one logical unit. After some tweaking of both the master XIP file and the ISG we were able to generate code for the entire system.

4. Next Steps
In order to have a working application, two final steps remain. First, aspects must be created and applied to the system. An aspect is essentially an XSL stylesheet containing the code necessary to implement the aspect in a system. Aspects looks for particular joinpoints in the code generation templates to determine where to place the various parts of the aspect code. Examples of joinpoints include “user-declare” where local variables can be declared, and “startup,” code that is executed when a pipe is initialized.

Our system requires several aspects for a working implementation. A database connection aspect would be needed to inject code needed to make an ODBC connection to a database on the network. Because databases have different connection properties, it may be useful to create some vendor-specific database aspects (for example, an aspect for Oracle, one for SQLServer, and one for DB2, etc.). A completed aspect could then be applied to any pipe requiring database access. Aspects could also be used for the multiplexer and data duplication pipes. This relatively simple aspect would simply copy the data from inports to outports based on the function of the pipe. Ideally this aspect would handle an arbitrary number of ports on either side.

The final step of implementation is to add any user-specific code. After generating the stubs and applying aspects, a bulk of the system will already exist. Here we see the power of the Infopipes architecture. The developer has only to fill in the core functionality of each component, without having to deal with plumbing issues.

5. Summary and Conclusions

We were able to successfully model an e-commerce application using the Infopipes toolkit and design methodology. The application incorporated a number of new concepts previously un-attempted using the Infopipes framework, including pipes for database access, multiplexing, data duplication, buffering, and routing. We see from the project’s outcome that the Infopipes architecture is well suited for a distributed e-commerce application and provides the infrastructure needed for quickly and effectively generating system code.

6. Works Cited

