iFlow: A Data Streaming Application Framework Based on a Uniform Abstraction

Andrew K. Lui, Mark W. Grigg, Michael J. Owen, T. Andrew Au
Defence Science and Technology Organisation
DSTO C3 Research Centre, Fern Hill Park, Canberra,
Department of Defence, ACT 2600, Australia
andrew.lui@computer.org; m.grigg@ieee.org

Extended Abstract

Many resource sensitive applications such as multimedia processing [1] and data-intensive computing [2] utilise data streams to pass information between sub-systems, components, processes and machines. The stream based programming model constrains applications to operate with smaller individual data blocks in a continuous manner. This reduces data storage and transport requirements, which in general leads to a more economical use of network and system resources. The creation of an application framework is highly desirable for the development of stream based applications. Application frameworks [3] can significantly ease development effort by providing proven architectures and well-engineered reusable components.

In the design of application frameworks, the representation of the target domain into appropriate abstractions has a strong influence on the framework characteristics. In particular, the level of abstraction is an important decision that controls the amount of detail hidden behind the abstractions. Generally a framework based on a lower level of abstraction results in a more fine-grained set of application components. This allows greater customisation in the applications. An example of a stream based programming framework is the CORBA A/V streaming specification [4]. This framework defines a set of components including multimedia devices, stream controls, flow controls, etc, for the management of data streams. However, the fine granularity of the components increases the complexity in application integration. Its use requires more detailed knowledge of the behaviour and characteristics of the components and their interactions (Figure 1).

This extended abstract presents a stream based programming framework called iFlow that provides a uniform set of components based on a higher-level abstraction of data streams. iFlow uses the universal abstraction of *pipes* to represent data streams. The real underlying form of the data streams is encapsulated, whether it is the transport, processing, generation, or termination of data. The pipes can therefore abstract away the complexities of data streaming such as machine boundaries, data processing methods, and stream I/O controls. With the implementation details hidden, the pipes offer a more straightforward application integration process. We argue that a uniform set of components not only eases application integration, but also eases the incorporation of adaptability into the applications. One form of adaptability is the on-demand capability to rapidly integrate components with suitable characteristics into applications. The uniform abstraction of the pipes supports the adaptability of applications through the addition, removal or replacement of pipes.

We have created a prototype implementation of the iFlow framework that supports the development of applications based on stream-based programming. The framework defines the pipe base class and the mechanism of pipe composition. The base class then allows the derivation of specialised pipes that can implement various data streams. A library of pre-fabricated, reusable and ready-to-use pipes is created to support application development. However, the more significant part of iFlow is the run-time support for the execution of pipes. The run-time support mainly deals with

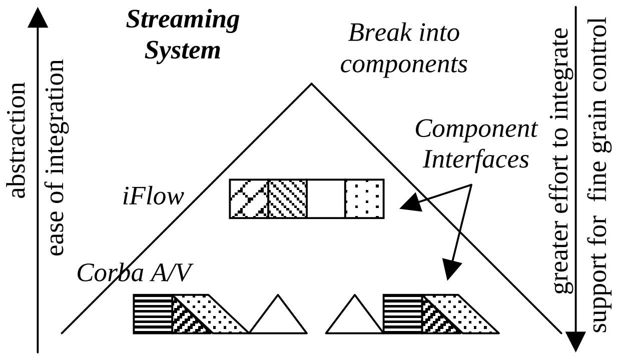

Figure 1. The effect of the level of abstraction on the support to adaptation and fine grain customisation. Component design at a higher abstraction can have a more uniform interface, facilitating integration and adaptation.

resource management that supports a performance guarantee in the flow characteristics of the data streams. The pipe abstraction exposes a common interface for resource specification and performance monitoring. Each pipe can offer particular flow characteristic guarantees. Resource management is coupled with pipe creation in the pipe factories, where external run-time support for quality of service (QoS) can be incorporated. Our prototype implementation has been developed using CORBA and Java on TCP/IP, however other types of middleware could also be employed.

We illustrate the basic capability of iFlow by applying it to an imagery dissemination application. The on-demand image dissemination is a very efficient method for viewing remotely located images across the network. The core idea is to request the optimal amount of data that is sufficient to satisfy the visual quality requirement, which in turn is based on the display resolution and the region of interest. Through an interactive viewer, users can manipulate the desired resolution and the region of interest. A number of wavelet based imagery formats support such on-demand dissemination. The stream based programming model is well matched to this application. We observe six different data streams in this application (Figure 2), and their respective pipe implementations are as follows: RequestInputPipe converts user interactions into a data stream of requests; MessagePipe moves the data stream of requests from the client machine to the server machine using sockets; RequestProcessorPipe processes the request stream and determines the required imagery data set to fulfil the requests; WaveletFileReaderPipe retrieves and streams the required imagery data from an image file; SocketPipe moves the data stream of imagery data from the server machine back to the client machine; and ReceiverPipe stores the imagery data into an image cache, which is then decoded for viewing. The imagery dissemination application is generated by first creating the pipes, joining them together, and

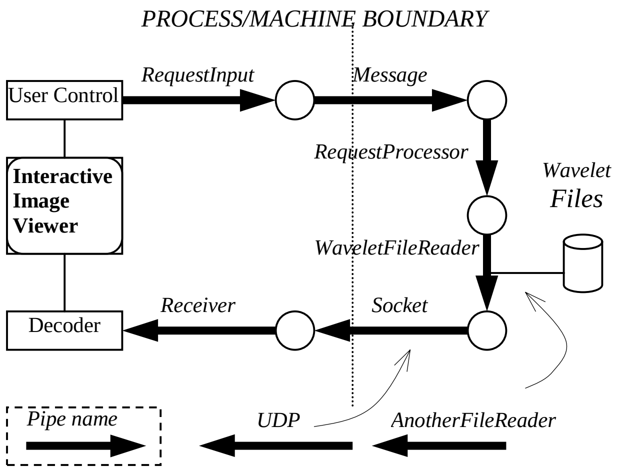

Figure 2. Building an image dissemination application with iFlow. Six pipes are used in the application to support interactive progressive image viewing. The pipe abstraction allows easy modification of the application by replacement, e.g. the UDPPipe replacing the SocketPipe.

activating the pipes. The uniform pipe abstraction makes it easy to modify the functionality of the application. For example, consider the case where the application proactively streams imagery data according to some intelligent prediction of future requests. We can implement an ImagePushPipe that determines the required imagery data according to the intelligent prediction. The new pipe is then simply placed along side the RequestProcessorPipe so that both the pipes will stream the desired imagery data requests into the WaveletFileReaderPipe.

The uniform pipe abstraction is also well suited to the incorporation of adaptive behaviour. As shown previously, iFlow defines a highly flexible application integration pattern. Pipes with the appropriate characteristics can be easily incorporated into applications. Adaptation can occur in two ways: the flow characteristics of a given pipe can be modified or a pipe can be completely replaced by another pipe. We use the above imagery dissemination application to illustrate adaptability under changing network conditions. For the first case, we can take advantage of an increase in bandwidth through re-negotiation with the pipe factory of the SocketPipe transfer-rate QoS. For the latter case, the SocketPipe is replaced by a UDPPipe, which performs better with high latency communication channels such as satellite or radio links [5].

We conclude that using a higher level of abstraction in designing a stream based programming framework can ease application integration and facilitate the incorporation of adaptability. We have presented a framework called iFlow that uses the uniform abstraction based on the concept of pipes. The use of iFlow in an adaptable imagery dissemination application has also been illustrated.

References

[1] S. Rixner et. al., “A Bandwidth-Efficient Architecture for Media Processing”, *Proceedings. 31st Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE Comput. Soc. 1998, pp.3-13. Los Alamitos, CA, USA.

[2] M. Benyon, T. Kurc, A. Sussman, and J. Saltz, “Design of a framework for data-intensive wide-area applications,” *Proceedings of 9th Heterogeneous Computing Workshop*, Los Alamitos, CA, USA, 2000.

[3] R. Johnson and B. Foote, “Designing Reusable Classes,” *Journal of Object Oriented Programming*, vol. 1, pp. 22-35, 1988.

[4] OMG, *CORBA Telecoms: Telecommunications Domain Specifications*, 1998.

[5] R. Prandolini, T. A. Au, A. K. Lui, M. J. Owen, and M. W. Grigg, “Use of UDP for Efficient Image Dissemination,” *Proceedings of SPIE Conference on Visual Communication and Image Processing (VCIP 2000)*, Perth, 2000.