Information Dissemination using Network Streams

Harry Behrens

Dept. of Information Engineering
University of Tokyo

Hidehiko Tanaka

Dept. of Information Engineering
University of Tokyo

Abstract

Network Streams are a generic means by which distributed applications can exchange information. Using Network Streams information, irrelevant of its type, can be exchanged using a generic interface. Information is encapsulated into classes, that are self-descriptive and partially reflective. Network streams act as generic container channels through which such objects are passed. Filter Agents are independent applications using Network Streams as their I/O paradigm. Because they are autonomous, they can travel to a remote site, connect to one of its applications and send filtered output to the agent's owner.

1 Introduction

The last years have brought dramatic development in the field of network based applications. More and more corporations, research institutions and individuals rely on network technology to manage their information flow.

Simple client/server applications, where clients retrieve data or use resources through a dedicated server machine have become to simplistic a model to model modern information based organizations.

The Internet with the underlying TCP/IP protocol suite inherently implies a peer-to-peer topology, where independent machines act as clients or servers depending on the context.

Based on these premises it has been our aim to define an architecture of distributed intelligent systems, that carry out intelligent information processing tasks. Assuming that these systems can be constructed, they would be immensely helpful in

- automatic information distribution,
- semantic search tasks,
- automatic SW execution and update,
- intelligent network management etc.

In order to facilitate the definition and implementation of such large scale distributed intelligent systems, a programming environment needs to be defined that lends itself to the construction of distributed intelligent agents and distributed knowledge bases.

This was the main motivation in creating the programming environment presented in this paper.

2 Basic Architecture

2.1 Motivation and Background

The main concept behind network streams is that it should be possible for applications running on different host machines on different networks to communicate and be used as building blocks to create large distributed systems.

On a lower level BSD sockets offer a sort of standard generic interface. In fact most of modern application level network protocols (SMTP, ftp, HTTP etc.) are based on sockets, allowing applications on different computer platforms to communicate through one standard interface.

The main drawback of this very powerful programming interface, is that it is relatively low level. Sockets are essentially handled like file handles and offer only simple read() or write() functionality.

Network streams - which are based on sockets, btw. - bring the level of abstraction higher while trying to keep the system as open and platform-independent as possible.

2.2 From Multimedia to Multiclass

One of the main reasons for the success of the WWW is that it handles multimedia in an intuitive and transparent way. What exactly does this mean on a more technical level? Well: files that get transferred from an HTTP server [1] to an HTML client [2] are sent with MIME [3] information, specifying the type of data transmitted. These MIME types are described using a two-level hierarchy specifying type and subtype (e.g. text/html or application/ms-word). Modern WWW browsers either handle these type directly (such as image/gif) or can be configured to invoke *helper applications* or pass the data to *plug-ins* that can process them.

It seemed a logical step to us to extend this very simple yet powerful mechanism to the programming task at hand. Instead of simply tagging data with a standardized type descriptor, we decided to use an object-oriented approach, where all data is transmitted through **Component Objects**. Component objects contain

- a descriptor, containing information about the class itself,
- data where applicable,
- method handlers

The descriptor component is standardized across all different classes. This means an application receiving an object can access at least the descriptor to determine whether it wants to handle the object or discard it.

2.3 Combining streams and objects

Combining the two basic ideas described in section 2.1 and section 2.2 resulted in the following model:

Any application that is to participate in the network stream environment needs to expose a set of input or output streams to the world. These streams are named and therefor accessible on a symbolic level. Each application also exposes a minimum set of externally callable management routines that allow third party scripting applications to create connections between stream endpoints.

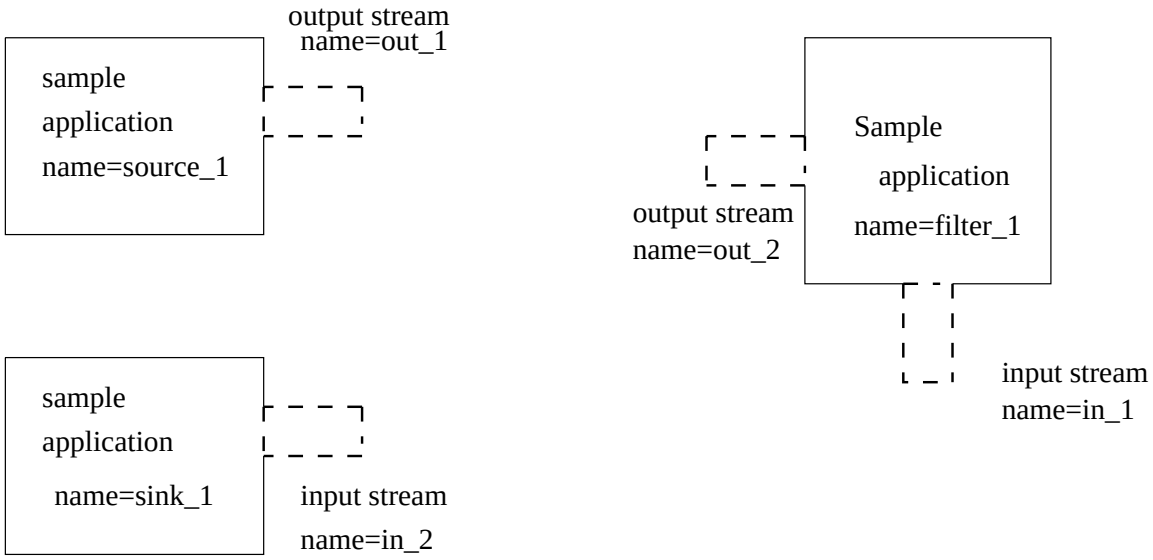

Figure 1: Individual applications with interfaces

The previous sketch shows three applications - all possibly residing on different sites - and their stream interfaces.

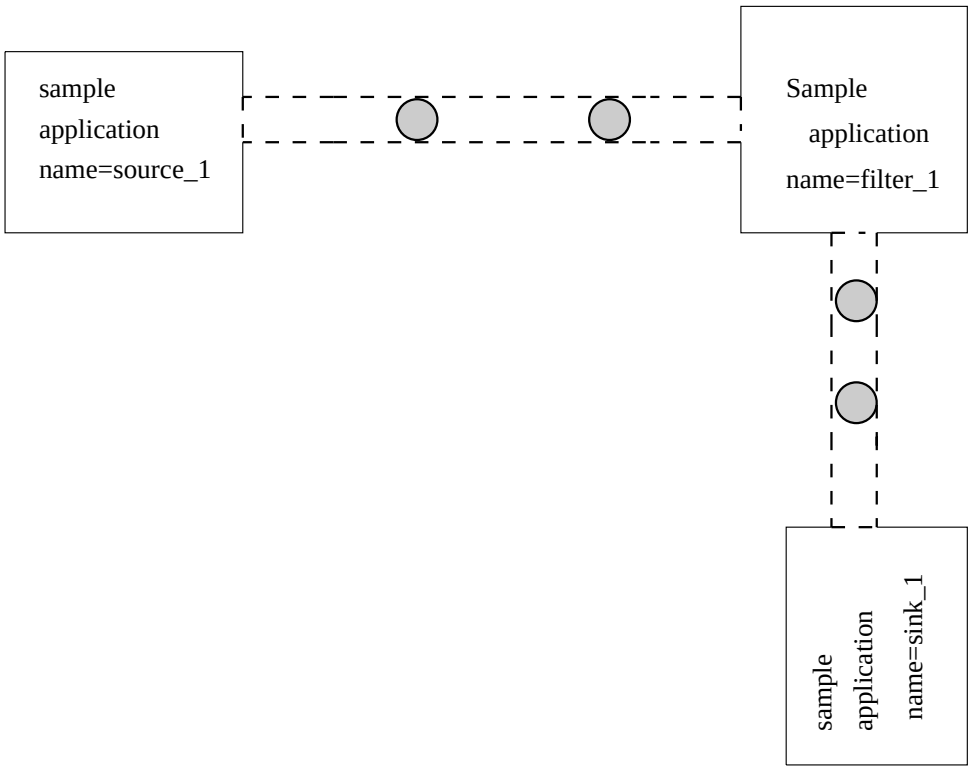

Figure 2: Connected System

This is a fully connected system with the shaded circles depicting objects being transferred.

3 Implementational issues

One of the main points in the system is, that applications need to expose an interface that allows the scripting components to dynamically interconnect the streams of various applications.

This has been achieved by encapsulating the remote configuration API in a library, which all applications need to link in. Through this API, an applications announces its presence and its interface description to a *site manager* upon startup. The site manager has a corresponding API library linked in, that allows it to access the stream interface of the applications it manages.

All requests for connection with an application thus have to go through the site manager.

Site managers and their locations are managed through a global name resolution system.

As far as the whole system is concerned an application only exists through its interfaces. Local input/output or other side effects (e.g. for logging purposes) may exist, but are the responsibility of the applications owner and are invisible to the system.

3.1 System architecture and components

The main components of the system are:

- **Remote configuration libraries:** these encapsulate the intrinsics of the remote invocations involved in remotely configuring a running application's I/O interface.
- **Site manager:** The site manager acts as a gateway between the network and a set of centrally administered applications. It is responsible for the servicing of requests and for the execution of the actual remote configuration.
- **Name Resolution Services:** NRSs exist on two levels:
  1. Site level: Sites need to be able to find each other's locations.
  2. Class level: All implementation information for the Component Object Classes can be retrieved over the network. This is important to allow transparent dynamic update of object behavior and implementation.
- **Scripting Environment:** The scripting environment reads in script files containing directives to create networked applications. It queries the system to verify the existence of the applications and sites addressed and then sends the necessary configuration messages to the site managers involved.

4 Discussion

The system introduced constitutes the programming environment upon which we hope to implement a high-level knowledge sharing system. Various such systems are at the moment topics of active research, the most notable being the DARPA *Knowledge Sharing effort* [4].

The actual implementation introduced in this paper tries to combine ease of use with platform independence and high level of abstraction. In the field of object oriented programming the *Common Object Broker Request Architecture* [5] tries to tackle the issue by making objects accessible over a network of ORBs. This approach thus tackles the issue at the object rather than the application level.

We believe that the system described is very easy to use and intuitive. Based on the reflective components of the Component Objects we will add semantic information in the future which will allow the system to systematically query site managers about the capabilities of its applications on a more semantic level.

The role of the site managers which is that of a passive *switchboard* at the moment, will be expanded to handle mobile applications (aka agents) in the future.

References

[1] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. *IETF Internet Draft <draft-ietf-http-v10-spec-04.html>*, October 1995.

[2] T. Berners-Lee and D. Connoly. Hypertext markup language - 2.0. *IETF RFC 1866*, November 1995.

[3] Borenstein N and Freed N. MIME (multipurpose internet mail extensions) part one:mechanisms for specifying and describing the format of internet message bodies. *IETF RFC 1521*, September 1993.

[4] Patil R., Fikes R., Patel-Schneider P., McKay D., Finin T., Gruber T., and Neches R. The DARPA knowledge sharing effort: Progress report. In Charles Rich and Bernhard Nebel, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*. Morgan Kaufmann, 1992.

[5] The Object Management Group. The Common Object Request Broker: Architecture and Specification. *Revision 1.1*, December 1991.