A Socket-Based Manifestation of Streams

Marc A. Criley
Quadrus Corporation, Huntsville, AL
adamac95@earthlink.net

Introduction

The Ada.Streams package, hereafter referred to simply as the Streams package, introduced into Ada 95 a standard mechanism for the storage and transmission of heterogeneous data within and amongst Ada programs. Section 13.13 of the Ada 95 LRM defines the streams features of the language, specifying the programmatic interface and behavior of this capability. It states that a "stream type may be implemented in various ways, such as an external sequential file, an internal buffer, or network channel." The Ada standard then goes on to define in Appendix A.12 the packages Streams.Stream_IO, Text_IO.Text_Streams, and Wide_Text_IO.Text_Streams, which provide stream-oriented operations on files.

Files, however, are merely one option for the implementation of stream-based communication. By suitably extending the Stream package's Root_Stream_Type and providing corresponding Read and Write procedures, a stream can be associated with any communication or storage medium. Applications employing the standard stream-oriented attributes defined in LRM 13.13.2 can transparently communicate, store, or load data items as governed by the application's extension of a stream.

This paper illustrates and implements a simple socket-based extension of a stream to demonstrate the flexibility of the Ada 95 Stream capability and the ease with which streams can be associated with other communication media.

Extending the Stream

The root type for Ada's streams is defined in the Streams package as:

```
type Root_Stream_Type is abstract tagged limited private;
```

Abstract Read and Write procedures are also defined:

```
procedure Read(Stream : in out Root_Stream_Type;
               Item   :    out Stream_Element_Array;
               Last   :    out Stream_Element_Offset) is abstract;


procedure Write(Stream : in out Root_Stream_Type;
                Item   : in     Stream_Element_Array) is abstract;
```

The first step in providing a socket-based manifestation of a stream is to extend the Root_Stream_Type as depicted in this full view of the type:

```
type Socket_Stream is new Ada.Streams.Root_Stream_Type with record
   -- Socket connection
   Socket : Sockets.Socket_Fd;
end record;
```

(The complete source code listings of the socket extension package and a simple demonstration of its use accompany this paper. The Sockets package referenced in this extension and in subsequent software listings is part of Samual Tardieu's AdaSockets collection for Unix, available at http://www.infres.enst.fr/ANC/.)

Completing the extension requires only implementing suitable Read and Write procedures:

```
procedure Read
   (Stream : in out Socket_Stream;
    Item   :    out Ada.Streams.Stream_Element_Array;
    Last   :    out Ada.Streams.Stream_Element_Offset) is
begin
   -- Read stream elements off the socket and set
   -- Last accordingly
   Sockets.Receive(Stream.Socket, Item);
   Last := Item'Last;
exception
   when Sockets.Connection_Closed =>
      raise Connection_Closed;
end Read;

procedure Write
   (Stream : in out Socket_Stream;
    Item   : in     Ada.Streams.Stream_Element_Array) is
begin
   -- Write the stream elements to the socket
   Sockets.Send(Stream.Socket, Item);
exception
   when Sockets.Connection_Closed =>
      raise Connection_Closed;
end Write;
```

As far as the implementation of the stream type for socket-based communication is concerned, the implementation is now complete. The only software still needed is some for managing the sockets themselves.

What Just Happened?

An application program invoking one of a type's stream-oriented output attributes, 'Write or 'Output, initiates the conversion of the data item into an array of Stream_Elements (a Stream_Element_Array), which is then passed as an argument to one or more dispatching calls of the corresponding stream's Write procedure. Streaming an elementary type results in a single dispatching call, while composite types result in a succession of calls as specified by LRM 13.13.2(9).[1]

Invoking a stream input attribute, 'Read or 'Input, similarly results in one or more dispatching invocations of the appropriate Read procedure, again with the specific number and sequence of invocations depending upon the composition of the type. The result of these invocations is, of course, the reconstruction of an instance of that type retrieved from the sequence of stream

1 The Ada developer must ensure that a multi-tasking program’s interaction with the stream's communication or storage medium—such as a socket—is protected. Because composite types are processed with a sequence of invocations, task switches may occur between successive calls if permitted by the act of interfacing to that communication or storage medium.

elements.

Beyond Sockets

The processing performed by the overriding Read and Write procedures must obviously ensure that any stream of elements written to some medium can be successfully read back in to reconstruct a valid instance of that type. Other than that, these procedures may employ any useful storage or communication medium, with the data needed to support that medium and the required processing provided in the extension of the Root_Stream_Type.

While this paper provides simple socket-based procedures doing nothing more than reading and writing to an associated socket, applications are free to employ more sophisticated techniques such as buffering, logging, multi-cast, etc.

Ada 95's Streams capability provides a robust and flexible foundation for simplifying the transmission and storage of heterogeneous data, accomplishing this in a straightforward manner that remains safe, easy to extend, and therefore tailorable to the requirements of the system.

Author's Addendum

As this paper was being prepared and the code verified against the latest release of AdaSockets, it was observed that the collection itself now provides a socket-oriented stream package, Sockets.Stream_IO. The implementation of that package is nearly identical to the one presented in this paper, confirming just how natural and straightforward extending Ada's stream capabilities can be.

```
with Sockets;
with Ada.Streams;

package Socket_Stream_Transport is

   -------------------------------------------------------------
   -- Connection was externally closed
   Connection_Closed : exception;

   -------------------------------------------------------------
   -- Extension of the Stream type to interact with sockets
   type Socket_Stream is new Ada.Streams.Root_Stream_Type with private;

   type Socket_Stream_Handle is access all Socket_Stream;

   -- Concrete implementation of abstract subprograms for
   -- converting arbitrary data items to Stream_Element_Arrays
   -- and reading and writing them through the Transport package.

   procedure Read
             (Stream : in out Socket_Stream;
              Item   :    out Ada.Streams.Stream_Element_Array;
              Last   :    out Ada.Streams.Stream_Element_Offset);
   pragma Inline(Read);

   procedure Write
             (Stream : in out Socket_Stream;
              Item   : in     Ada.Streams.Stream_Element_Array);
   pragma Inline(Write);

   -------------------------------------------------------------

   -- Associate a socket with a stream
   procedure Set_Transport (Stream : in out Socket_Stream;
                            Socket : in     Sockets.Socket_Fd);

   -- Return the socket associated with a stream
   function Get_Transport (Stream : Socket_Stream)
                            return Sockets.Socket_Fd;

private
   type Socket_Stream is new Ada.Streams.Root_Stream_Type with record
      -- Socket connection
      Socket : Sockets.Socket_Fd;
   end record;

end Socket_Stream_Transport;
```

Listing 1, Socket_Stream_Transport.ads

```
with Ada.Streams;
with Sockets;

package body Socket_Stream_Transport is
```

```
   -----------------------

   procedure Set_Transport (Stream : in out Socket_Stream;
                            Socket : in     Sockets.Socket_Fd) is
   begin
      Stream.Socket := Socket;
   end Set_Transport;

   -----------------------

   function Get_Transport (Stream : Socket_Stream)
                             return Sockets.Socket_Fd is
   begin
      return Stream.Socket;
   end Get_Transport;

   -----------------------

   procedure Read(Stream : in out Socket_Stream;
                  Item   :    out Ada.Streams.Stream_Element_Array;
                  Last   :    out Ada.Streams.Stream_Element_Offset) is
   begin
      -- Read stream elements off the socket and set Last
      -- accordingly
      Sockets.Receive(Stream.Socket, Item);
      Last := Item'Last;

   exception
      when Sockets.Connection_Closed =>
         raise Connection_Closed;
   end Read;

   -----------------------

   procedure Write(Stream : in out Socket_Stream;
                   Item   : in     Ada.Streams.Stream_Element_Array) is
   begin
      -- Write the stream elements to the socket
      Sockets.Send(Stream.Socket, Item);

   exception
      when Sockets.Connection_Closed =>
         raise Connection_Closed;
   end Write;

   -----------------------

end Socket_Stream_Transport;
```

Listing 2, Socket_Stream_Transport.adb

```
-- A simple package for setting up a socket-based stream server.

with Ada.Streams;
with Socket_Stream_Transport;
```

```
package Socket_Stream_Server is

   -- Open the socket for connections
   procedure Create_Server
     (Port   : in Natural;
      Server : in Socket_Stream_Transport.Socket_Stream_Handle);

   -- Wait for a connection on the socket
   procedure Await_Connection
     (Server : in Socket_Stream_Transport.Socket_Stream_Handle;
      Client : in Socket_Stream_Transport.Socket_Stream_Handle);

   -- Send the given data through the socket stream
   procedure Send
     (Client : in Socket_Stream_Transport.Socket_Stream_Handle;
      Data   : in Ada.Streams.Stream_Element_Array);

   -- Read enough data off the socket to fill the data item
   procedure Receive
     (From_Client : in     Socket_Stream_Transport.Socket_Stream_Handle;
      Data        :    out Ada.Streams.Stream_Element_Array);

   -- Close the connection
   procedure Close
     (Connection : in Socket_Stream_Transport.Socket_Stream_Handle);

end Socket_Stream_Server;
```

Listing 3, Socket_Stream_Server.ads

```
with Ada.Exceptions;
with Ada.Streams;
with Sockets;

with Text_IO; use Text_IO;

package body Socket_Stream_Server is

   use Socket_Stream_Transport;

   ----------
   -- Send --
   ----------

   procedure Send
     (Client : in Socket_Stream_Transport.Socket_Stream_Handle;
      Data   : in Ada.Streams.Stream_Element_Array)
   is
   begin
      -- Now send the data
      Sockets.Send(Get_Transport(Client.all), Data);
   end Send;

   -------------------
```

```
-- Create_Server --
-------------------

procedure Create_Server
  (Port   : in Natural;
   Server : in Socket_Stream_Transport.Socket_Stream_Handle) is

   use Sockets;

   Server_Socket : Sockets.Socket_FD;

begin
   Socket (Server_Socket, AF_INET, SOCK_STREAM);
   Setsockopt (Server_Socket, SOL_SOCKET, SO_REUSEADDR, 1);
   Bind (Server_Socket, Port);
   Listen (Server_Socket);
   Set_Transport(Server.all, Server_Socket);
end Create_Server;

----------------------
-- Await_Connection --
----------------------

-- Wait for a connection on the socket
procedure Await_Connection
  (Server : in Socket_Stream_Transport.Socket_Stream_Handle;
   Client : in Socket_Stream_Transport.Socket_Stream_Handle) is

   Client_Socket : Sockets.Socket_FD;

begin
   loop
      Put_Line ("Waiting for new connection");
      Sockets.Accept_Socket
        (Get_Transport(Server.all), Client_Socket);
      Set_Transport(Client.all, Client_Socket);
      Put_Line ("New connection acknowledged");
      exit;
   end loop;
exception
   when E : others =>
      Put_Line("Exception raised: " &
               Ada.Exceptions.Exception_Name(E));
      raise;

end Await_Connection;

-------------
-- Receive --
-------------

-- Pull enough data off the socket to fill the data item
procedure Receive
  (From_Client : in     Socket_Stream_Transport.Socket_Stream_Handle;
   Data        :    out Ada.Streams.Stream_Element_Array) is

begin
```

```
      Sockets.Receive(Get_Transport(From_Client.all), Data);
   exception
      when Sockets.Connection_Closed =>
         raise Socket_Stream_Transport.Connection_Closed;
   end Receive;

   -----------
   -- Close --
   -----------

   procedure Close
     (Connection : in Socket_Stream_Transport.Socket_Stream_Handle) is

      Server_Socket : Sockets.Socket_FD;

   begin
      Server_Socket := Get_Transport(Connection.all);
      Sockets.Shutdown(Server_Socket);
   end Close;

end Socket_Stream_Server;
```

Listing 4, Socket_Stream_Server.adb

```
-- A minimal package that a client can use to connect and disconnect
-- socket-based streams

with Socket_Stream_Transport;

package Socket_Stream_Client is

   -- Exception thrown when a connection cannot be made
   Connection_Failed : exception;

   -- Connect to the specified server
   procedure Connect_To_Server
     (
      -- Host identifier on which a server is running
      Host   : in String;

      -- Port number on which the server is listening
      Port   : in Natural;

      -- Stream connected via socket
      Stream : in Socket_Stream_Transport.Socket_Stream_Handle);


   -- Break the connection
   procedure Disconnect_From_Server
     (Stream : in Socket_Stream_Transport.Socket_Stream_Handle);

end Socket_Stream_Client;
```

Listing 5, Socket_Stream_Client.ads

```
with Socket_Stream_Transport;
with Sockets;
use  Sockets;

package body Socket_Stream_Client is

   -----------------------------------------------------------

   -- Connect to the specified server
   procedure Connect_To_Server
     (
      -- Host identifier on which a server is running
      Host   : in String;

      -- Port number on which that server is listening
      Port   : in Natural;

      -- Stream connected via socket
      Stream : in Socket_Stream_Transport.Socket_Stream_Handle) is

      -- The socket to associate with a stream
      Stream_Socket : Sockets.Socket_FD;

   begin
      -- Create the socket
      Sockets.Socket (Stream_Socket, AF_INET, SOCK_STREAM);

      begin
         -- Make a single attempt at connection
         Sockets.Connect (Stream_Socket, Host, Port);

         -- Associate the connected socket with the stream
         Socket_Stream_Transport.Set_Transport
            (Stream.all, Stream_Socket);
      exception
         when Sockets.Connection_Refused =>
            -- Release the socket
            Sockets.Shutdown(Stream_Socket);
            raise Connection_Failed;
      end;

   exception
      when Constraint_Error =>
         -- Sockets.Socket raises a Constraint_Error if it cannot
         -- create a socket
         raise Connection_Failed;
   end Connect_To_Server;

   -----------------------------------------------------------

   -- Break the connection
   procedure Disconnect_From_Server
     (Stream : in Socket_Stream_Transport.Socket_Stream_Handle) is

      -- The socket associated with the stream to be closed
      Stream_Socket : Sockets.Socket_FD;
   begin
      -- Release the socket
      Stream_Socket :=
         Socket_Stream_Transport.Get_Transport(Stream.all);
      Sockets.Shutdown(Stream_Socket);
   end Disconnect_From_Server;

   -----------------------------------------------------------

end Socket_Stream_Client;
```

Listing 6, Socket_Stream_Client.adb

```
with Ada.Strings.Unbounded;
use  Ada.Strings.Unbounded;

package Data_Type is

   -- A type definition package providing a common type transmitted
   -- between clients and servers

   type Data_Priorities is (Low, Medium, High);

   type Data is record
      Sender   : Unbounded_String;
      Value    : Natural;
      Quality  : Float;
      Priority : Data_Priorities;
   end record;

end Data_Type;
```

Listing 7, Data_Type.ads

```
with Ada.Strings.Unbounded;
use  Ada.Strings.Unbounded;
with Ada.Text_IO;
use  Ada.Text_IO;
with Data_Type;
with Socket_Stream_Server;
with Socket_Stream_Transport;

procedure Server_Test is

   Server : Socket_Stream_Transport.Socket_Stream_Handle :=
     new Socket_Stream_Transport.Socket_Stream;

   Client : Socket_Stream_Transport.Socket_Stream_Handle :=
     new Socket_Stream_Transport.Socket_Stream;

   Data_Item : Data_Type.Data;

begin
   Put_Line("Creating the server socket");
   Socket_Stream_Server.Create_Server(5000, Server);
   Put_Line("Listening for a client");
   Socket_Stream_Server.Await_Connection(Server, Client);

   Put_Line("Client connected!");
   Data_Type.Data'Read(Client, Data_Item);

   Put_Line("Received data:");
   Put_Line("Sender   : " & To_String(Data_Item.Sender));
   Put_Line("Value    : " & Natural'Image(Data_Item.Value));
   Put_Line("Quality  : " & Float'Image(Data_Item.Quality));
   Put_Line("Priority : " & Data_Type.Data_Priorities'Image
                              (Data_Item.Priority));

   Put_Line("Closing client connection and server");
   Socket_Stream_Server.Close(Client);
   Socket_Stream_Server.Close(Server);

end Server_Test;
```

Listing 8, Server_Test.adb

```
with Ada.Strings.Unbounded;
use  Ada.Strings.Unbounded;
with Ada.Text_IO;
use  Ada.Text_IO;
with Data_Type;
with Socket_Stream_Transport;
with Socket_Stream_Client;
use  Socket_Stream_Client;

procedure Client_Test is

   Client : Socket_Stream_Transport.Socket_Stream_Handle :=
     new Socket_Stream_Transport.Socket_Stream;

   Data_Item : Data_Type.Data;

   Connected : Boolean;

begin
   Put_Line("Connecting to the server socket");

   begin
      Connect_To_Server("localhost", 5000, Client);
      Connected := True;
   exception
      when Connection_Failed =>
         Put_Line("Connection failed");
         Connected := False;
   end;

   if Connected then
      Put_Line("Connected, now streaming data...");
      Data_Item :=
        (Sender   =>
            To_Unbounded_String("This is your client speaking."),
         Value    => 95,
         Quality  => 1.0,
         Priority => Data_Type.High);
      Data_Type.Data'Write(Client, Data_Item);

      Put_Line("Data transmitted, disconnect");
      Disconnect_From_Server(Client);
   end if;

end Client_Test;
```

Listing 9, Client_Test.adb