Network Transfer of Control Data : An Application of the NIST SMART DATA FLOW

Martial MICHEL, Ph.D.
System Plus Inc., 1370 Piccard Drive - Suite 270
Rockville, MD 20878, USA
&
NIST, 100 Bureau Dr - MS 8940
Gaithersburg, MD 20899, USA
martial.michel@nist.gov

and

Vincent STANFORD
NIST
vincent.stanford@nist.gov

and

Olivier GALIBERT
NIST

ABSTRACT

Pervasive Computing environments range from basic mobile point of sale terminal systems, to rich Smart Spaces with many devices and sensors such as lapel microphones, audio and video sensor arrays and multiple interactive PDA acting as electronic brief cases, providing authentication, and user preference data to the environment. These systems present new challenges in distributed human-computer interfaces such as how to best use sensor streams, distribute interfaces across multiple devices, and dynamic network management as users come an go, and as devices are added or fail.

The NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY SMART DATA FLOW system is a low overhead, high bandwidth transport mechanism for standardized multi-modal data streams. It is designed to allow integration of multiple sensors with distributed processing needed for the sense-recognize-respond cycle of multi modal user interfaces. Its core is a server/client architecture, allowing clients to produce or subscribe to data flows, and supporting steps toward scalable processing, distributing the computing requirements among many network connected computers and pervasive devices.

This article introduces the communication broker and provides an example of an effective real time sensor fusion to track a speaker with a video camera using data captured from multi-channel microphone array.

Keywords: SMART DATA FLOW, MEETING ROOM, Data Transfer, Flows of Data, Data pipelining

1 INTRODUCTION

Real time multimedia information is often transfered as streaming video and audio. Pervasive Computing is an emerging field that, among other things, makes intensive use of several technologies to accomplish the integration of multi modal sensor streams. Although not limited to real time streaming data, the distributed processing of high volume of information does require a resilient and low overhead transport mechanism. In order to create a test bed for Pervasive Smart Spaces, the SMART DATA FLOW system (SDF) version 1 was released in 1999 by the NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) SMART SPACE project (SMS). It was designed to support integration of devices and sensors into processing steps distributed over multiple computing nodes sharing flows of data transparently. By abstracting the details of data transport, the SMART DATA FLOW simplify the development of complex, multi node, multi step distributed applications that are needed for smart environments.

This article presents the concepts of the NIST SDF, as well as its underlying component library. It also presents a detailed example of its use in a full scale application : a multi-channel audio data is used to localize a speaking person inside a smart room and point a camera at the estimated source bearing. Finally, we discuss the current problems and limitations of the current SDF and introduce new concepts for the next generation SDF-II that

will allow more dynamic data flow systems to be constructed.

2 THE NIST SMART DATA FLOW

Objectives & Needs

The NIST SMART DATA FLOW system was designed to enable research work in the field of Pervasive Computing and Smart Spaces, and help manage the numerous, easily accessible computing devices connected to each other in the ubiquitous network infrastructure they need to be effective. Its focus is on advanced forms of human computer interaction, integrating wireless networks with dynamic service discovery, automatic device configuration, and sensor based perceptual interfaces. The objectives are to facilitate :

- Identification of security mechanisms for privacy, integrity, and accessibility.
- Interconnection of components and systems to explore key issues in distributed smart spaces.
- Establishment of an integrated multi-sensor perceptual interface test bed for smart work spaces.
- Providing a multi-sensor data recording environment for the production of standard test and reference materials.

In order to support research on the effective design of Pervasive Smart Spaces, an abstract, high speed, low overhead, data transport mechanism allowing distributed processing of sensor data, was needed. Therefore, in order to ease the development of such spaces, we defined a simple system to work among distributed systems and allow network transfer of high data density *flow*s between *client*s interconnected by a *server* crossbar.

Concepts & Conception

The servers: The servers establish the initial crossbar between the different physical systems that will send and receive data flows. They also provide the push/pull mechanism that is used in flow transfers; with client flows being transported using the server established crossbar. In this architecture, the server must be started before clients can connect to the SDF. The crossbar is established by the server using a static of list the available physical nodes.

The clients: Clients are processes that produce flows, consume flows, or both. They can be thought of as small data processing functions using an abstract transport mechanism to receive inputs and send outputs; each SDF client should be limited to a specific function so that they can be used as building blocks for a higher level system functionalities.

Clients are not bound to a specific physical system, unless they require specific components only available on that system, such as capture clients which require data acquisition hardware, e.g: a multi channel audio capture board. Otherwise it is very simple to transfer a client application from one node to another. This is made possible by the naming convention for clients which is defined to uniquely refer to a client by the combination of its name and group, and not the system it is started on.

In order for a client to work with flows, it must attach itself to a server running on the local system and describe the flows it uses or produces to the server.

The Flows: Flows are 1-to-n producer/consumer system elements. They can be of a raw data type but the SDF library also defines some standard types for audio, video, vector, matrix and opaque structures.

A flow is also uniquely defined by its name and its group. The use of groups for both clients and flow allow a simple domain and sub-domain topology, which is useful in cases where each *Video Capture Client* on systems such as `camera1` to `camera5` produces instances of the same *Video In* flow, yet it needs to be able to distinguish each such flow. This is done by specifying a different group for each flow instance.

Note that the data-type also matters; a video flow cannot be attached to an audio flow, but the data-type is not the main mean of identifying flows.

Each flow is defined and made available to other clients with a fixed history size; it is possible for a client to request a buffer that is older than the most recent one, given that the server, which controls the flow memory management process, still possesses it.

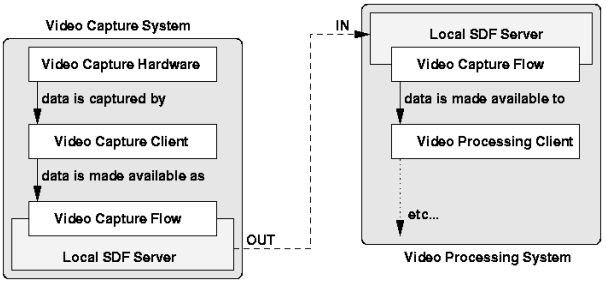

Figure 1. Flow mechanism

Data Transfer Using Flows: The figure 1 represents the internal mechanism underlying the capture of a video flow from a hardware sensor, here a camera coupled with a capture card, and how it is sent from the *Video Capture System* to the *Video Processing System* :

1. The current frame is captured by the *Video Capture Hardware* using the API provided for the capture card. This is done from the *Video Capture Client* application.
2. The *Video Capture Client* requests a send buffer from the *local SDF server* and copy the captured frame into this buffer. Then the client request for this buffer –now encapsulated inside a flow– to be made available to subscribers (the is the "send" process).
3. The *Local SDF Server* located on the *Video Capture*

System pushes the information to the *Local SDF Server* located on the *Video Processing System*.

4. The *Local SDF Server* on the *Video Processing System* pulls the information received via the transport mechanism (both clients can be local in which case the transport mechanism is shared memory or if separated by a network, a TCP/IP packet stream) and add it to its local buffers for this flow.
5. The *Video Processing Client* has made a blocking request to get the latest available buffer from the *Video Capture Flow*. The *Local SDF Server* makes it available to the client as a read-only information.
6. The *Video Processing Client* is now able to use the data contained inside the flow, but must release the buffer back to the *Local SDF Server* once finished.

Note that the *Video Processing System* can itself be a *flow* producer. In that case after processing the data contained in the received flow, it will itself request an output buffer to encapsulate its data into a flow, and *send* it (once again, the *Local SDF Server* will push the data to the *Local SDF Server* of the client subscribing to the produced flow).

Using Data Flows In Clients

```
Create a flow
Loop (
   Allocate a buffer
   Write data in the buffer
   Send the buffer
)
```

ALGORITHM **2.1. Opening a flow in emit mode**

In Emit Mode: The simplified algorithm presented on 2.1 gives an abstract view of the procedures involved in emitting a flow.

On the client side, the first thing to be done is to announce to the local server the flow that will be provided, including information about its name, group, type and sub-type. Then for each buffer to be sent, the client requests a buffer from the server, writes its data into this buffer, and asks the server to send this buffer.

The server does the counterpart job; when given the information about all the flows provided by the client, it propagates the information among the crossbar to all other servers who will in turn check if they know subscribers for this flow. Then for each buffer to be sent, the server provides, upon request, a buffer to the client, and when this buffer is returned (asked to be sent by the client), the server makes it available to all subscribing clients by sending it to their respective local server.

In Receive Mode: The reception algorithm shown on 2.2 presents the steps involved in the reception of a flow.

```
Subscribe to the flow
Loop (
   Read the buffer
   Process its data
   Release the buffer
)
```

ALGORITHM **2.2. Subscribing to a flow in receive mode**

The client first tells the local server about the flow it is waiting to subscribe to, providing its full name as well as the data types and format it requires. Then for each buffer to be received, the client does a blocking wait on this flow, read the content of the pushed buffer, processes the received data (making a local copy of the data if it needs to make changes[1]), and releases the buffer to the server.

During the initialization process, the server gets the information from the new client about the flows it requires. If the server has such a flow registered from another client it contacts the remote server providing the client emitting this flow and establishes the client subscription to it.

If the flow is not yet available, it will provide it to the receiving client once an emitting client produces it. If the sub-type of the data requested by the client is different from the sub-type proposed for subscription by the remote server, the local server will perform the conversions needed to provide the proper buffers to its client if this can be done. For example, the remote client sends 24 bits JPEG compressed frames and the local client requires 24 bits uncompressed frames; the server will uncompress the source frame into the destination format. Then, for each buffer received from the emitting-client via its local server, the server makes it available to local clients, after appropriate conversions, and provide on explicit request.

Once the client has finished its processing, the server memory manager will decrement some usage counter for this buffer. When the usage count drops to zero the memory space is made available for re-use.

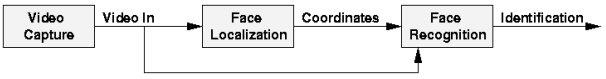

Figure 2. A simple flow graph example

A Simple Face Identification Example:A simple example of the pipelining and branching of data flows is presented in figure 2 where the *Video Capture* client provides its *Video In* flow to both the *Face Localization* client and the *Face Recognition* client.

[1]this is to make sure that if more than one client on the local server need this flow, the data will not be altered

The flows provide data driven process control so that for each video frame transported, the localization client provides the recognition client with the coordinates of a face box for the corresponding video frame. The local SDF servers manage the flow storage queues to allow the processes to proceed at their own pace, and even out scheduling jitter and differences in process execution times. As long as the slowest process keeps up with the real time video capture source, the graph process all paths without losing frames.

The face recognition client uses the localization coordinates to apply recognition algorithms, such as Eigenface to the appropriate portion of each video frame to classify the face as that of a person from its template library. This information is then provided in an *Identification* flow, which is time tagged for later analysis.

3 PERSON TRACKING USING SOUND LOCALIZATION

In this section we present a real scenario used in the NIST Smart Space SDF test bed. It describes how to steer a camera based on source bearing estimates using phased array processing techniques and the NIST Mark-II microphone array. The source bearing allows the camera to frame a speaker, so that additional video processing, such as face localization can be applied.

Hardware involved

The NIST SMART SPACE Laboratory is composed of the NIST SMART DATA FLOW , multiple sensors and data acquisition hardware. Some of these include :

- A PC based video capture system (here an AMD K6 300Mhz with 128MB of memory) running Linux and an *Linux Media Labs LML33* video capture card, based on the Zoran ZR36057 chipset, connected to a Sony EVI D30 camera with VISCA capabilities.
- A PC based sound capture system (here an AMD K6 450Mhz with 256MB of memory) also running Linux and a PCI card with eight multiplexed 16 bit ADCs capturing 64 channels from the NIST Mark-II 64 channel microphone array. The data channels are sampled at approximately 22050Hz[2].
- Our computation nodes are based on Athlon XP2000+ system with 1.5GB DDR.

Phased Array Processing Algorithm for Source Bearing

This is a uniformly spaced line array with a 1.5 cm inter sensor separation supporting acquisition of frequencies up to approximately 11 kHz without spatial aliasing. To facilitate camera pointing, it is placed at the horizontal center of the microphone array and is steered coaxially with the estimated source bearing. This spacing of the sensors is designed to allow the identification of multiple sources, as well as the most dominant source, using simple delay-and-sum beam forming. In order to find the dominant source we simply compute a field of 61 beams, each steered three degrees from its neighbor by introducing appropriate delays at each array element, and compute the incident power on each beam. We then scan the resulting energy versus direction map for the peak energy, and impute the dominant source bearing to that direction. In order to improve resolution we highpass filter the individual sensors to block energy with wavelengths too long to be steered out by this array. Using numerous closely spaced sensors allows this computation to proceed in real time using modest computational resources owing to its simplicity. More advanced analysis might allow multiple sources to be resolved, possibly allowing exclusion of simultaneous speakers.

Detail of the Flows involved

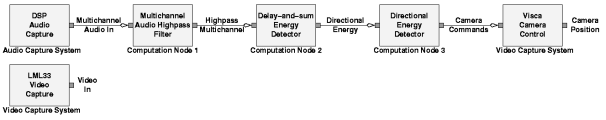

Figure 3. Flows involved in Sound based Localization

Since the computation and data acquisition steps in this process are beyond the abilities of a single commodity PC, we used the SDF to distribute the captured data to other nodes in a pipeline, which together compose the source bearing estimation and camera steering process. As shown in figure 3, the entire process is pipelined into five distributed steps :

1. On the *Audio Capture System*, the *DSP Audio Capture* client collects data from the microphone array and creates a *Multichannel Audio In* flow that is made available for subscription to other clients.
2. On *Computation Node 1*, the *Multichannel Audio Highpass filter* subscribes to the multichannel flow, and applies a high pass filter to each of the 64 channels. The client emits a *Highpass Multichannel* flow for subscription.
3. *Computation Node 2* uses the highpass flow for delay-and-sum beamforming of 61 beams. This information is encapsulated inside the *Directional Energy* flow.
4. *Computation Node 3* uses directional energy flow to find the peak direction which indicates the angle of the speaker relative to the microphone array. The client then uses source bearing angle to generate a *Camera Commands* flow.
5. Finally, the *Video Capture System* uses the previous flow to generate camera commands and sends them to the hardware (here a VISCA enabled Sony EVI D30 camera) to slew the camera to frame the

[2]The Mark-III NIST microphone array is in development, and will have lower parts cost, improved 24 bit ADCs, better signal-to-noise ratios, and an on board FPGA to handle data framing and direct connection to the SDF via Ethernet UDP packets for flexible deployment

person speaking. Note that this client also generates a *Camera Position* flow that could be combined with other data to support person identification using face recognition, or other features for gesture recognition.

It is important to remark that the computation nodes 1, 2 and 3 can transparently be distinct systems, or if the processing power allows it, a single system. The clients in the pipeline do not need to be aware of the hardware configuration that executes them.

The Real Application

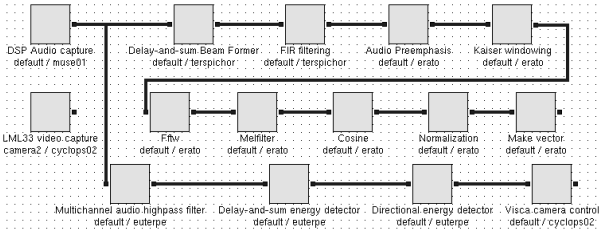

Figure 4. Screenshot of actual GUI of the SMART DATA FLOW control center for Sound based Localization

Figure 5. Screenshot of the face localization and tracking result

Figure 4 shows the actual client map for the process presented above as well as additional processing for spoken language processing. Two branches are represented on this map. The first, shown across the bottom part of the figure, implements the client steps presented for the source bearing estimation of a speaker. The second branch, making a serpentine path from the top to the middle, implements a multi step pipeline for the computation of a Mel Frequency Cepstrum from a beamformed source. The Finite Impulse Response (FIR) filtering provides signal conditioning to remove the low frequency noise present in most buildings. Next is the Cepstrum, a feature extraction algorithm for speech signals commonly used for speech recognition and speaker identification. The processing steps include preemphasis filtering to flatten the spectrum, time domain windowing to improve spectral resolution, Fast Fourier Transform, Mel frequency filter bank computation, Cosine transform and Cepstral mean normalization to reduce the impact of stationary background noise, and to equalize the channel. The result can be seen on the screenshot in figure 5 that displays the view as seen by the camera with the speaker centered in the frame, and includes the box for face localization, as discussed previously.

4 FUTURE OF THE NIST SMART DATA FLOW

Our plans for enhanced functionality in SDF version 2, are designed to mitigate the limitations of version 1. We have used version 1 to generate terabyte scale meeting databases with over two hundred microphones and five simultaneous camera views, which has shown it to be a scalable and robust distributed processing system. However, this was on a fixed network topology using about fifteen separate nodes generating about a gigabyte of data per minute and a private switch. However, Pervasive Computing environments require support for dynamic devices and service discovery, real time fault tolerance, and some level of self configuration.

Limitations of the NIST SMART DATA FLOW I

Some of the current issues we feel need to be addressed in the present version of the SDF are as follows :

- All local SDF servers must be started before any client can be instantiated. It is impossible to dynamically add servers to the currently running list, or to run a client on a system without a local SDF server already running.
- The servers maintain the crossbar, but they have no safeguard mechanism; if a server fails, the crossbar is broken, causing all servers to shutdown. Clients, which must be attached to a running local SDF server will therefore also fail.
- The servers also convert data-types, where possible, so that flows can be used by clients requiring it in different forms; for example, up-converting 16 bits audio to 24 bits if needed. This adds overhead in the server, and greater risk of failure. If a conversion is faulty, it may force servers to exit in error, disconnecting its clients and breaking the crossbar, and so killing the entire flow graph.
- Each flow buffer for a local client is sent/received by the local server. So the clients do no direct transfer among themselves : all flows and control communications are handled by the server crossbar, adding to the server latency.

SMART DATA FLOW **II**

Design of the NIST SMART DATA FLOW II is currently under way, with some of the features to correct the above limitations under consideration :

- Dynamic clients addition to a current running SDF flow graphs. A local server will be started by each

local client. This mechanism will make it possible to insure that servers for required systems are started, but needs a server discovery mechanism at server start up that allows the new ones to yield to preexisting ones if they are present. Clients will therefore now be responsible for the server start up; and in case of server failures, a new one will be restarted by the local client(s). The newly created server will re-establish the previous server crossbar.

- The flow communications will now be handled by a client-to-client direct connection; the server will only maintain the crossbar and send special control information from server-to-server. A side effect of this is that data-type conversion will be handled by the clients in data object methods, possibly with conversion thread or a new hidden client for this purpose, either on the producer or consumer system, depending on the number of subscribers to the converted flow. An extension of this idea shall be a limited load balancing; if both clients' systems are heavily loaded, a third client located on another system can handle conversions.
- Removing the static server dependencies will allow interfaces with mobile PDAs as they arrive, and wireless networks. Clients, used on devices, will be able to come into the crossbar and leave at any time.

5 Conclusion

This paper detailed the requirements for a tool such as the SDF in the context of Pervasive Computing and Smart Spaces, presenting its conceptual basis and a practical use of its flow components on a sensor fusion test case. Low communication communication overhead is crucial to real time response in dynamic multi modal environment processing sensor data streams. Therefore, the communication broker must support high bandwidth use, mostly limited by the speed of the network and the computation speed of its nodes rather than in heavy weight protocols.

When the SDF II is operational, its new architecture and improved failure recovery will allow a more practical use of its capabilities for pervasive handheld devices, and mobile user interaction. PDAs will then be able to come and go with their users, and re-integrate it transparently when they return.

We are currently forming a Pervasive Computing Standards working group, and interested parties are encouraged to contact the authors by e-mail at NIST.

Acknowledgement

The work described in this paper was collaboratively conceptualized and developed by the authors and many other members of the NIST Smart Space Project. However, we wish to acknowledge Olivier Galibert as the author of a substantial majority of the core data flow software during his time at the Smart Space Project. His work on the project is gratefully appreciated.

HTTP REFERENCES

- NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY : `http://www.nist.gov/`
- SMART SPACE : `http://www.nist.gov/smartspace/`
- MEETING ROOM : `http://www.nist.gov/speech/test_beds/mr_proj/`
- SYSTEM PLUS, INC. : `http://www.sysplus.com/`