Sharing Data in Collaborative, Interactive Performances: the SenseWorld DataNetwork

Marije A.J. Baalman, Harry C. Smoak, Christopher L. Salter
Design and Computation Arts
Concordia University
Montréal, QC, Canada
`majbaalm,hsmoak,chrissal@alcor.concordia.ca`

Joseph Malloch, Marcelo Wanderley
Input Devices and Music Interaction Lab
McGill University
Montréal, QC, Canada

Abstract

The SenseWorld DataNetwork framework addresses the issue of sharing and manipulating multiple data streams among different media systems in a heterogenous interactive performance environment. It is intended to facilitate the creation, rehearsal process and performance practice of collaborative interactive media art works, by making the sharing of data (from sensors or internal processes) between collaborators easier, faster and more flexible.

Keywords: Data exchange, collaborative performance, interactive performance, interactive art works, sensor data, OpenSoundControl, SuperCollider, Max/MSP

1. Introduction and Background

The SenseWorld Data Network addresses one of the major challenges in the research/creation of interactive live performance work: the sharing and manipulation of raw and/or conditioned sensor data among different media systems (real time audio and video, lighting, mechatronics, show control, etc). While the introduction of common data sharing protocols like Open Sound Control (OSC) [1] has facilitated communication between disparate software environments, such protocols still do not address how individual human collaborators work with real time data at the *local client level* in specific media and application domains.

Establishing interactive relationships utilizing sensor data to manipulate and shape light, sound and image requires an understanding of the unique spatio-temporal characteristics of the individual media themselves. Simultaneously, the creation of a complex performance project in a rehearsal context demands techniques that provide designers - individually and collectively - with a shared set of nomenclatures and tools for the extemporaneous manipulation of real time data in order to create sequences or events that have experiential affect for an audience.

Media designers often must access and process sensor data using the programming conventions and embedded conceptual frameworks of their particular software and hardware platform. Moreover, they usually discuss the use of such data only within such frameworks and its associated techniques. Even in instances where a common software platform (e.g., Max/MSP+Jitter) that can natively accommodate multiple media types (video, audio, etc.) is available and shared among all designers, a higher level framework must still be provided for shared data communication and manipulation.

Our aim in developing the SenseWorld DataNetwork is to support multiple different practices, without requiring each designer to conform to the idiosyncrasies of any technical implementation within a particular media practice. The framework is intended to support *coordinated* collaboration with real time data within a live interactive performance context, in order to facilitate the creation of specific performance events involving multiple types of media.

2. Design Criteria

During our work on previous interactive media performance projects (such as *Schwelle* [2]), where data was shared between three computers to manage incoming input from body and environment-based sensors, audio and lighting control, we relied on OSC for lower level communication of data between the software environments SuperCollider [3] and Max/MSP [4]. As we moved on to other collaborative projects involving multi-channel real time sensor data, we identified a need for higher level framework which was extensible over multiple projects and could make data communication and processing easier, faster and more reliable.

The final design criteria were:

- Any client should be able to subscribe to (receive) data
- Any client should be able to supply (send) data
- Restore configuration quickly
- Usable within heterogeneous media software environments
- Enable collaboration between heterogeneous design practices
- Enable efficiency of collaboration within the limited timeframe of rehearsals

2.1. Related work

The *KeyWorx* [1] framework [5] is one of the few frameworks that addresses some of these issues, but it seems to have an emphasis on net-based art and collaborative projects between different locations, rather than collaboration between different media environments within one location.

The McGill Digital Orchestra [6] mapper tools propose an alternative network based framework, but focus on musical application and have slightly different implementation decisions. The aim there is to facilitate a plug and play approach for plugging in sensor based interfaces and have them communicate with digital (musical) instruments. The users of this network do not necessarily have to be the developers of the sensor interfaces or digital instruments themselves, and not necessarily be programmers of some kind.

Our goal here is to facilitate communication between software environments that artists already use for interactive performance works. Thus, we assume that the users are familiar with their own environments and medium. Also, we do not enable communication to take place at the device level, but rather at the software level, though different clients can share the data they retrieve from sensor devices.

3. The SenseWorld DataNetwork framework

The framework's core is implemented in SuperCollider (SC), which is available as open source software and runs on several platforms (Linux, OSX, Windows and FreeBSD). Clients have been implemented in SC and Max/MSP so far. The OSC namespace is well defined, so it should be trivial to implement clients for other software environments such as PureData [2] or Processing [3] .

What follows is a technical description of the implementation of the framework. Users of the framework need not be familiar with the inner workings of the framework (or have experience in programming SC), so long as their software environment supports OSC and is able to comply with the OSC namespace conventions in order to communicate with the network. Thus, the framework is designed to allow for ease of use within a designer's own creative practice.

A central host receives all data messages and manages the client connections (see figure 1). Each client can subscribe to one or more data *nodes* in order to use that node's data in its own internal processes. Furthermore, each client can publish data onto the network by creating a node. A new client can query the network concerning which nodes are present and is informed when new nodes appear after the client has been registered. Thus, a *data node* can be understood as a collection of data that belongs together, e.g., data coming from the same sensor device or the output of a particular device such as the DMX control stream for theatrical light.

[1] `http://www.keyworx.org`
[2] `http://www.puredata.info`
[3] `http://www.processing.org`

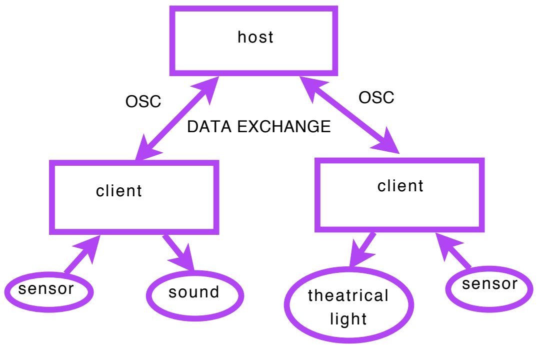


Figure 1. Diagram of the SenseWorld DataNetwork structure.

Within each node there are *slots* which represent single data values, for example, a data node representing a 3-axis accelerometer has three slots, with each slot corresponding to one axis. If a client is only interested in one slot of a node, he can subscribe specifically to that slot.

3.1. OSC implementation

The network is accessed through an OSC interface [4] , which allows a client to join the data network, access its data and also create its own data nodes on the network.

The general setup is as follows: a client first sends a registration message to the data network server. The client will immediately begin receiving *ping* messages to which it must reply with *pong* messages confirming the client's availability. Following the initial registration, the client can submit a query message in order to receive a complete list of nodes and slots currently available from the network. The client can then subscribe to selected nodes and slots, and subsequently will receive data from the nodes and slots it is subscribed to via data messages corresponding to the subscribed data sources.

The client can supply a new node to the network by using the `/set/data` message (which is also used subsequently to set new data). A client can also label the nodes and slots it has created. Whenever a new node or slot is added (by any client) or changed (e.g., when it gets a label), the client will receive a new info message automatically. All messages to the server have a reply, which is either the requested info, a confirmation message or a warning or error.

In comparison, the Digital Orchestra tools provide a decentralized network using one general multicast port on the network to settle the ports and namespaces of clients. Since not all interactive media software environments support listening to multicast channels, we elected not to use this approach. Rather our assumption is that each client will settle its listening port itself within the operating system of the computer on which it runs. The host can then distinguish

[4] assumed to be used via the UDP layer, though TCP can be used as well if the client supports it. The OSC protocol in itself is not dependent on the underlying protocol, but is implemented on top of TCP and/or UDP in most software systems

each client by its IP address (automatically included in the OSC message) and a port which is an argument of each OSC message in the namespace. The latter is necessary as several clients (Max/MSP among them) do not send OSC messages from the same port as they are listening to.

3.2. Auto-recovery

Practical lessons derived from rehearsal and performance experiences reminds us that software applications and processes can be unexpectedly and fatally interrupted (i.e., crash). For this reason, a fast and automatic recovery of all previously instantiated connections is critical. The following methods are implemented to enable fast and automated recovery in such situations. Following (re)start of the host server and (re)establishment with the network, an announce message is broadcast on several ports. In addition, the server updates a publicly readable file with the current active listening port. Moreover, the host can restore previously connected clients from information stored in a server readable file. In turn, the client has read access over the network to the host configuration file and automatically retrieves the port information to which it has to register. Further, the client implements an auto-configuration process triggered upon receipt of a host announce message and a response from the host indicating the client has successfully registered.

3.3. SuperCollider implementation

The SuperCollider (host) implementation is done via a set of custom written classes:

SWDataNetwork base class for the network.
SWDataNode base class for a data node.
SWDataSlot base class for a data slot.
SWDataNetworkSpec implements the labelling of the nodes and slots of the network.
SWDataNetworkOSC implements the OSC interface
SWDataNetworkOSCClient keeps track of a connected client

Data nodes have both unique IDs (integer numbers) and human readable labels (e.g., node “3” has the label “accelerometer”). Data slots are automatically numbered, according to the order in which they appear, as they are set in the network; they can also be given a label. The labelling is not done automatically, so that the naming becomes a conscious and integral part of establishing shared nomenclatures for the collaboration. This encourages consideration by the users of what the data represents and its potential use. The label specification (the “spec”) can be stored between sessions so it can be recalled again upon startup.

Each data node and slot has methods to print debugging messages, set an action to be performed upon new incoming data, scale and/or remap the data and create a control bus on the audio server[5] with the data. Each data node also can specify an action to be taken in case there has been no input to the node for a certain amount of time (e.g., trying to reconnect to an external device).

If a client creates a node, that node is linked to the client (the client becomes the “setter” of the node), and no other client can set data to that node. Client configuration can also be stored to a file and be used for recovery on startup.

All data from the network can be written to a log-file in a text format containing lines for each time step with tab-separated data values. The log can be opened and played back with the class **SWDataNetworkLog**.

Finally, a graphical user interface[6] has been implemented to monitor the status of the data network and the state of each node.

3.3.1. The client

The class **SWDataNetworkClient** implements an OSC client to the SenseWorld DataNetwork so that an external SC client can also be part of the network. It implements the client side of the OSC interface. It inherits from the class **SWDataNetwork** to manage the data it receives, and thus has the same interface in its use within SC.

In a setup with several collaborators using SC, one of them will act as a host for the network, while the other SC participants register as clients to that host. Since the code interface for both is almost the same, apart from the initialisation, it is very easy for SC participants to prepare and test their own inputs and outputs to the DataNetwork individually, and switch to the shared network during collective rehearsals.

3.3.2. Derived data

Deriving data from existing nodes can be done for example by combining data from various nodes, calculating statistical properties of data, or smoothing data.

To facilitate this, we have developed methods to do this either making use of the language features of SC, or by making use of the server’s unit generators to perform these calculations.

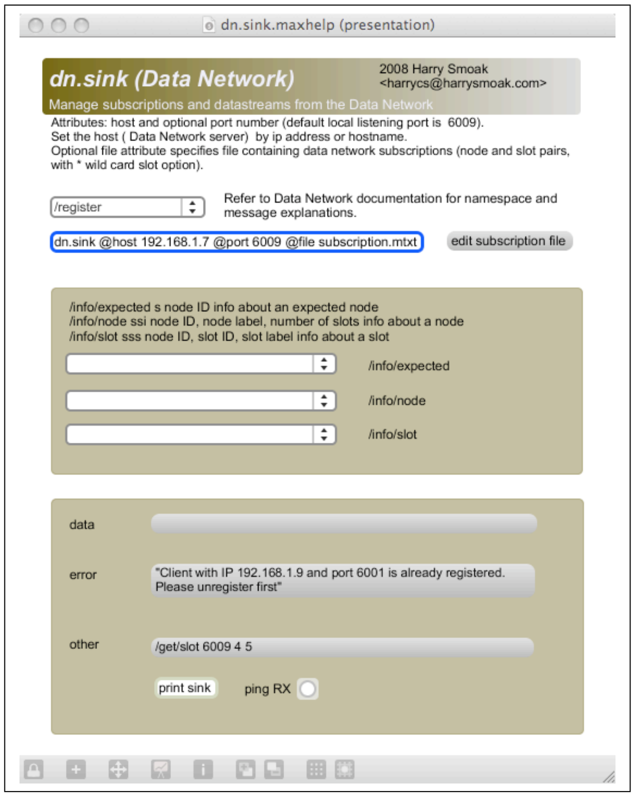
3.3.3. Example

Here we give a short example of how to use the network in a local configuration:

```
x = SWDataNetwork.new; // define the network
// get data input from incoming serial data
// (from an XBee network)
q = XBeeSMS.new( "/dev/ttyUSB0", 115200 );
// the action sets the data that comes in to the network:
q.action_({ |msg| x.setData( msg[0], msg.copyToEnd(1) ); });
// start the parsing of the input data:
q.start;
// we expect floor pressure sensor data on node 2
x.addExpected( 2, \floor ); // adds a label
// we want to create a node with a measure of the
// sample variance on node 102 and give it label \floorVar
x.addExpected( 102, \floorVar );
// the data comes in as 8bit integers, so we scale it
x[\floor].scale = 1.0/255;
```

[5] SC consists of two parts: *sclang*, which is the programming language, and *scsynth*, which is a dedicated audio server.

[6] for which unfortunately the space is lacking for a screenshot.



2008 Harry Smoak
<harrycs@harrysmoak.com>

Manage subscriptions and datastreams from the Data Network

Attributes: host and optional port number (default local listening port is 6009).
Set the host (Data Network server) by ip address or hostname.
Optional file attribute specifies file containing data network subscriptions (node and slot pairs, with * wild card slot option).

Refer to Data Network documentation for namespace and message explanations.

dn.sink @host 192.168.1.7 @port 6009 @file subscription.mtxt

/info/expected s node ID info about an expected node
/info/node ssi node ID, node label, number of slots info about a node
/info/slot sss node ID, slot ID, slot label info about a slot

"Client with IP 192.168.1.9 and port 6001 is already registered. Please unregister first"

/get/slot 6009 4 5

Figure 2. The DataNetwork sink Max/MSP patch.

```
// we create a bus for the data (s is the audio server):
x[\floor].createBus( s );
// we create the sample variance node:
~floorVar = StdDevNode.new( 102, x, x.at( \floor ).bus, s );
~floorVar.start;
// now we have raw data available on node 2, x[\floor]
// and sample variance data on node 102,  x[\floorVar]
```

3.4. Max/MSP client

A collection of abstractions is implemented for Max 5 and Max 4.5/4.6. The collection consists essentially of two abstractions: (1) dn.sink for managing the data network connection, subscriptions, registration, queries, and get requests, and (2) dn.source for adding and publishing data streams to the node server. A sample patch, help file, and text-based configuration file are included for each. The text file based configuration allows for portability, ease of editing (by external script or editor), and quick startup and recovery. Support for network announce and info messages is also implemented, allowing for autoconnection and real time updates of data network changes.

4. Usage and Further Development

We are currently using the SenseWorld DataNetwork framework in the dance production *Chronotopia* with the Attakkalari Centre for Movement [7], which toured in India in February 2009. In this production the network is used to distribute camera motion tracking data, audio feature extraction data, light output data and frame timing to two collaborators, one controlling a CCFL light matrix and audio playback from SC and another controlling video from Max/MSP.

Further development of the network includes making derived data available on the network through the OSC interface. While SC provides endless possibilities to derive data, a common set should be requestable through the network. The protocol for this still needs to be determined and should be easily extensible for additions by various users.

The SenseWorld DataNetwork is one part of a suite of tools being developed for live performance contexts between Concordia University and McGill University's IDMIL.

[7] http://www.attakkalari.org

5. Acknowledgments

This work was supported by grants from the Social Sciences and Humanities Research Council of Canada and the Hexagram Institute for Research/Creation in Media Arts and Sciences, Montréal, QC, Canada.

5.1. Download

The SenseWorld DataNetwork is available from http://quarks.sourceforge.net, or through SuperCollider's built-in Quarks extension manager. It is released as open source software under the GNU/General Public License.

References

[1] M. Wright, A. Freed, and A. Momeni. OpenSoundControl: State of the art 2003. In *2003 International Conference on New Interfaces for Musical Expression, McGill University, Montreal, Canada 22-24 May 2003, Proceedings*, pages 153–160, 2003.

[2] Marije A.J. Baalman, Daniel Moody-Grigsby, and Christopher L. Salter. Schwelle: Sensor augmented, adaptive sound design for live theater performance. In *Proceedings of NIME 2007 New Interfaces for Musical Expression, New York, NY, USA*, 2007.

[3] J. McCartney. Supercollider - environment and programming language for real time audio synthesis and algorithmic composition. http://www.audiosynth.com, http://supercollider.sourceforge.net.

[4] Max/MSP programming environment. http://www.cycling74.com/products/maxmsp.html.

[5] Sher Doruff. Collaborative praxis: The making of the keyworx platform. In Joke Brouwer, Arjen Mulder, and Anne Nigten, editors, *aRt&D: Research and Development in the Arts*. V2/NAI Publishers, Rotterdam, 2005.

[6] Joseph Malloch, Stephen Sinclair, and Marcelo M. Wanderley. A network-based framework for collaborative development and performance of digital musical instruments. In Richard Kronland-Martinet, Solvi Ystad, and Kristoffer Jensen, editors, *Computer Music Modeling and Retrieval. Sense of Sounds: 4th International Symposium, CMMR 2007, Copenhagen, Denmark, August 2007, Revised Papers*, number ISBN 978-3540850342 in Lecture Notes in Computer Science. Springer, 2008.