Sense/Stage — low cost, open source wireless sensor and data sharing infrastructure for live performance and interactive realtime environments

Marije A.J. BAALMAN
Harry C. SMOAK
Vincent DE BELLEVAL
Brett BERGMANN
Christopher L. SALTER
Design and Computation Arts
Concordia University
Montréal, Québec
Canada,
marije@nescivi.nl and sensestage@gmail.com

Joseph MALLOCH
Joseph THIBODEAU
Marcelo M. WANDERLEY
Input Devices and Music Interaction Lab
McGill University
Montréal, Québec
Canada,
joseph.malloch@gmail.com

Abstract

SenseStage is a research-creation project to develop a wireless sensor network infrastructure for live performance and interactive, real-time environments. The project is motivated by the economic and technical constraints of live performance contexts and the lack of existing tools for artistic work with wireless sensing platforms. The development is situated within professional artistic contexts and tested in real world scenarios.

In this paper we discuss our choice of wireless platform, the design of the hardware and firmware for the wireless nodes, and the software integration of the wireless platform with popular media programming environments by means of a data sharing network, as well as evaluation and dissemination of the technology through workshops. Finally, we elaborate on the application of the hardware and software infrastructure in professional artistic projects: two dance performances, two media projects involving environmental data and an interactive, multisensory installation.

Keywords

Wireless sensing, mesh networks, data sharing, realtime performance, interactive environments.

1 Introduction

SenseStage is a research-creation project to develop small, low cost and low power wireless sensor hardware together with software infrastructure specifically for use in live theater, dance and music performance as well as for the design of interactive, real-time environments involving distributed, heterogeneous sensing modalities. The project consists of three components:

- a series of small, battery powered wireless PCBs that can acquire and transmit input from a range of analog and digital sensors,
- an open source software environment that enables the real-time sharing of such sensor data among designers and
- plug in modules that enable the analysis of such sensor data streams in order to provide building blocks for the generation of complex dynamics for output media.

The project emerged from a desire to address a novel, emerging research field: distributed, wireless sensing networks for real-time composition using many forms of output media including sound, video, lighting, mechatronic and actuation devices and similar.

Three specific factors have motivated the SenseStage project:

1) *Economic and technical constraints of live performance:* There is an increasing interest in the use of sensing technology in live performance contexts. The economic and cultural constraints of live performance, however, make the integration and use of such technologies difficult. It is seldom possible to have long rehearsal periods with full access to a technical setup that is equivalent to the eventual performance space due to the industrial model of cultural production — show in, show out – leaving no room for exploration of new technological possibilities and the artistic impact. By providing a solution for easy application of the technology, the short rehearsal periods and "techweeks" can be spent more on the artistic exploration of the technology, rather than solving technological problems.

2) *Lack of tools for artistic use:* There are many groups currently researching the applications of wireless sensing networks, but their design decisions are normally motivated by en-

gineering innovations, thus leading to efficient yet, prohibitively expensive and complex systems. Also, despite the large number of research projects in the field, there are few wireless sensing platforms that are actually available for real world use, or that are affordable for artists. In addition these wireless sensing platforms rarely integrate with the software tools that artists use for making music, sound and media.

3) *Real world testing scenarios:* Much of the research agenda for the project was driven by many years of artistic work and technological development of tools to facilitate the creation of interactive performances and installations. These events employed distributed sensing and mapped such input data to complex parameter spaces for the control of sound and other media in real-time (e.g. *Schwelle* [Baalman et al., 2007] and *TGarden*[Ryan and Salter, 2003]). A key design element of the SenseStage project is thus to deploy SenseStage technologies into real world, professionally driven testing environments to see how such tools function "in the wild" and outside of the standard lab, demo-driven mode normally given to the presentation of new technologies.

2 Hardware

For the hardware wireless sensing node for Sense/Stage, our main requirements were cost per unit — since low costs allow experimentation with large numbers of nodes — and immediate availability. We investigated several options for wireless nodes developed by other groups, such as the μParts[1] [Beigl et al., 2005], the EcoMote[2] [Park and Chou,], the Tyndall Motes[3] and the Intel Motes[4][Nachman et al., 2005], but none of these satisfied our needs, or more importantly they were simply not available.

Our design goals were:

- Low cost
- Small form factor
- Flexible sensor configuration
- Usable for control of motors, LEDs, and other actuators.
- Operable in large groups (10+ nodes)
- Long battery life
- Ease of use
- Programmable, so that the board can take care of more logic and processing of data, if desired by the user

We decided to use the XBee in combination with the Arduino[5] platform, as the XBee already provided us with the needed ad-hoc network structure. Additionally, several other developers have documented their experience using XBees in conjunction with Arduinos[6] allowing us to skip some common development pitfalls.

We based our board design on the Arduino Mini Pro, being able to tap into many available firmware libraries, as well as the development and programming environment. Furthermore, Arduino is widely used in open source, artistic, physical computing contexts, so our board will be easy to use for this community.

The main focus then was to design a PCB that was small, and to develop standard firmware that makes it easy to setup and use the boards, as well as exploring the use of and integration with the XBee wireless chips.

The PCB layout of the SenseStage MiniBee is shown in figure 1. The first board revision came to a unit cost price of about 32 CAD, excluding the XBee chip, for a manufacturing run of 100 boards (PCB creation, assembly and parts). With a larger manufacturing run and allowing for a longer assembly time, this per unit cost will drop considerably.

2.1 Firmware

The firmware is a collection of functions to handle wireless transmission and communication, sensor reading and basic read/write operation on any available pin of the MiniBee. The firmware is built using our own library for the Arduino environment which allows users to quickly build their own application for the MiniBee.

Currently the following sensors/actuators are supported by the firmware:

- Analog sensors (connected to the analog input pins, e.g. resistive sensors, analog accelerometers, infrared distance sensors)

[1] `http://particle.teco.edu/upart/`

[2] `http://www.ecomote.net/`

[3] `http://www.tyndall.ie/mai/WirelessSensorNetworksPrototypingPlatform_25mm.htm`

[4] `http://techresearch.intel.com/articles/Exploratory/1503.htm`

[5] `http://www.arduino.cc`

[6] e.g. the ArduinoXbeeShield`http://www.arduino.cc/en/Main/ArduinoXbeeShield`, the Arduino Xbee Interface Circuit (AXIC) `http://132.208.118.245/~vitamin/tof/AXIC/` and the Blushing Boy MicroBee R3 `http://blushingboy.org/content/microbee-r3`

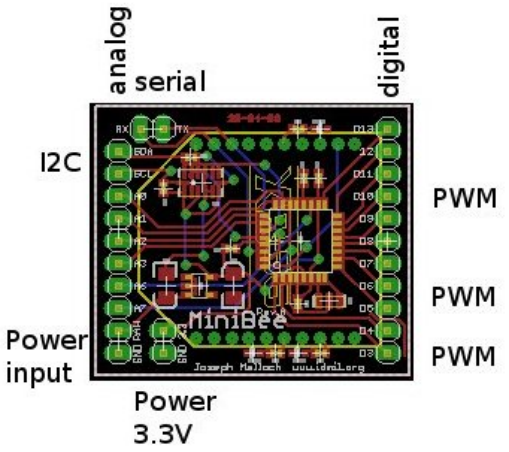

Figure 1: The SenseStage MiniBee PCB, rev. A. The XBee is mounted on the other side of the board. Changes in the second revision include smaller board-size and the footprint for a coin-cell.

- Digital sensors (on/off, e.g. buttons and switches)
- LIS302DL accelerometer[7], using I2C
- Relative humidity and temperature sensor[8]
- Ultrasound sensors[9]
- PWM output (e.g. dimmable LEDs, motors)
- Digital output (on/off)

The serial protocol is loosely based on the Serial Line Internet Protocol (SLIP)[10], and is set up as simply as possible to ensure that data packets are small. A regular package is built up as follows:

- escape character, followed by message type
- message ID
- node ID
- data bytes
- delimiter character

A full reference of the different messages supported is given in table 1.

The firmware is configured through a host computer which allows to quickly change its operation without having to physically reprogram the microcontroller. This approach is not unlike Firmata [Steiner, 2009] with the difference that our firmware stores its latest configuration in the EEPROM of the MiniBee.

Each time the MiniBee boots up, it reads the serial number of the attached XBee[11] and relays this information to the coordinator node connected to the host computer. The host then assigns a unique node ID to the MiniBee and optionally sends a new configuration to the MiniBee. If no new configuration is received, the MiniBee can access its configuration through its EEPROM. The host computer software remembers the known boards and XBee serial numbers so node IDs and configurations are maintained for a project.

Using an 8 MHz clock on the board, the maximum baud rate that can be achieved is 19200 baud. In a next revision we will include a faster (up to 20MHz) crystal, which will allow the use of higher baudrates.

Future work also includes writing a wireless bootloader to fully reprogram the SenseStage MiniBee without the need to manipulate the boards (see for example LadyAda[12]). The extra components needed to do this will be added in the next revision of the board.

3 Software

In order to make the data from the wireless sensor nodes available to several collaborators on a project simultaneously, we developed the SenseWorld DataNetwork. It is intended to facilitate the creation, rehearsal and performance of collaborative interactive media art works, by making the sharing of data (from sensors or internal processes) between collaborators easy, fast and flexible. Our aim is to support multiple media practices and allow different practitioners to use the software to which they are accustomed. The framework is intended to support *coordinated collaboration* with real-time data and multiple media types within a live interactive performance context.

The framework is different from the *KeyWorx*[13] framework [Doruff, 2005], which emphasizes net-based art and collaborative projects between different locations, and the McGill Digital Orchestra Tools[14] [Malloch et

[7]There is a footprint on the board for this sensor.

[8]Sensirion SHT1x series

[9]The "Ultrasonic Ranger", `http://www.robot-electronics.co.uk/htm/srf05tech.htm`

[10]`http://tools.ietf.org/html/rfc1055`

[11]using the AT command mode. Other people have made an Arduino library for communicating with XBees in API mode. See `http://code.google.com/p/xbee-arduino/`

[12]`http://www.ladyada.net/make/xbee/arduino.html`

[13]`http://www.keyworx.org`

[14]There is a Max/MSP bridge between the SenseWorld DataNetwork and the Digital Orchestra Tools so they

description	type	data	sender
Announce	'A'		server
Quit	'Q'		server
Serial number	's'	Serial High (SH) + Serial Low (SL)	node
ID assignment	'I'	msg ID + SH + SL + node ID + (*config ID*)	server
Configuration	'C'	configuration bytes	server
PWM	'P'	node ID + msg ID + 6 values	server
Digital out	'D'	node ID + msg ID + 11 values	server
Data	'd'	node ID + msg ID + N values	node

Table 1: Message protocol between host and MiniBee nodes. Type is preceded by the escape character (92), and messages are delimited with the delimiter character (10). The escape character is used to escape to the message type, and whenever the delimiter, the escape character, or the carriage return character (13) occurs in any of the other bytes. General convention for the message type: server message in upper case, node message lower case. (*...*) indicates an optional byte.

al., 2008], which primarily focus on the mapping and performance of monolithic digital musical instruments.

The final design criteria were to:

- Tight integration with the wireless sensing platform
- Allow reception of data from any node by any client (subscription)
- Allow transmission of data to any node by any client[15] (publication)
- Restore network and node configuration quickly
- Be usable within heterogeneous media software environments
- Enable collaboration between heterogeneous design practices
- Enable efficiency of collaboration within the limited timeframe of rehearsals

3.1 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, PureData, Max/MSP, Processing, Java, and C++ so far. The OSC namespace is well defined, so it should be trivial to implement clients for other software environments.

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 2). 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.

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. 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.

3.2 Integration with MiniBee mesh network

We have specifically integrated the connection to the SenseStage MiniBee network, by providing options to configure, send and receive data from the sensor network, through the host of the DataNetwork.

The host communicates with the wireless network through a serial protocol. It manages all

can be used together.

[15] only one client can set data to a specific node at a time.

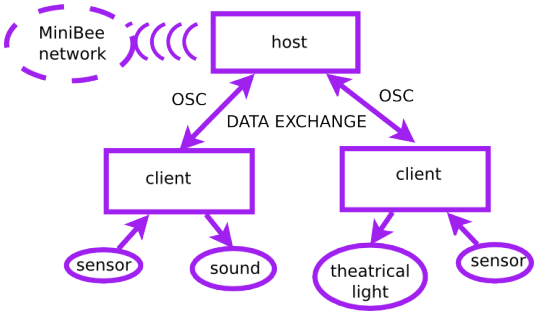

Figure 2: Diagram of the SenseWorld DataNetwork structure.

the incoming and outgoing data from and to the wireless nodes. The data from a MiniBee will appear onto the network as a DataNode, while each attached sensor is a *data slot* in this *data node*. Clients can then subscribe to this *node* to receive its data. Furthermore, a client can send a message to map the data that is on a *data node* the client has created to the digital outputs or pulse width modulation outputs of a MiniBee, in order to control, for example, lights or motors.

3.3 OSC implementation

The network is accessed through an OSC interface[16], 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 presence. 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 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, or cannot configure the port they are sending from.

3.4 Auto-recovery

Practical lessons derived from rehearsal and performance experiences remind 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.

3.5 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 net-

[16] assumed to be used via the UDP layer. 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

work.

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[17] 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 has been implemented to monitor the status of the data network and the state of each node (see figure 3 for the client's version of this GUI).

3.5.1 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

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

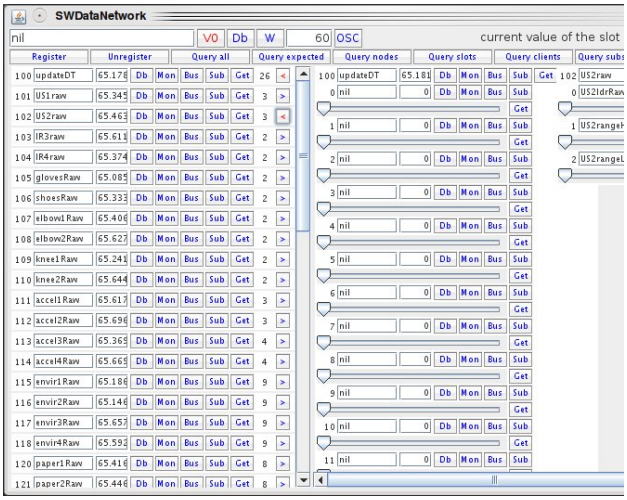

Figure 3: The DataNetwork SC Client interface.

the data it receives, and thus has the same interface in its use within SC, with some additional methods for interaction with the host.

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. In figure 4 an example is given of the code interface to both the host and the client.

3.5.2 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.

3.6 PureData and Max/MSP clients

Two abstractions provide access to the SenseWorld DataNetwork within a Pd or Max patch: **dn.node** and **dn.makenode**. Both implementations require a private variable for the host IP address and for the client name (shared between all instances on a client), but otherwise take care of the details of talking to the network.

dn.node subscribes to an existing node, outputting the received data as a list. In Pure Data each instance subscribes to a single node, whereas in Max it is possible to receive data from multiple nodes using a single **dn.node** ob-

Example of setting up the host in SuperCollider:

```
// create an instance of the DataNetwork
x = SWDataNetwork.new;
// adds the host OSC interface
x.addOSCInterface;
// create an instance of the class
// managing the MiniBees:
q = SWMiniHive.new( x );
// make GUIs for the DataNetwork and the hive:
x.makeGui;
q.makeGui;
```

An example of using the client in SuperCollider:

```
// create a DataNetwork client with the name ‘‘sc’’:
x = SWDataNetworkClient.new(‘‘192.168.0.104’’,‘‘sc’’);
// see what is in the network
x.queryAll;
// we know that node 2 has interesting data coming
// from a floor pressure sensor, so we subscribe:
x.subscribeNode( 2 );
// we want to create a node with a measure of the
// sample variance on node 102 and give a label
x.addExpected( 102, \floorVar );
// node 2 has the label \floor to access it
// we create a bus for the data (s is default server):
x[\floor].createBus( s );
// we create the sample variance node:
~floorVar = StdDevNode.new( 102, x, x[\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]
```

Figure 4: The DataNetwork SC code interface.

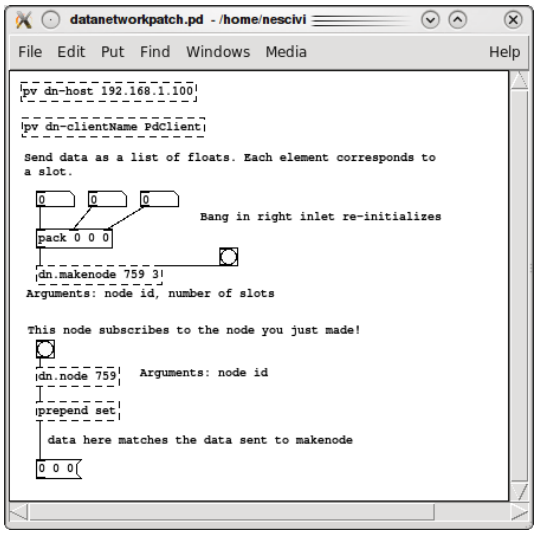

Figure 5: Screenshot of the PureData client.

ject (in which case each node's data list is preceded by its node ID).

dn.makenode does the opposite, publishing data to the network using a unique node ID. The data must be formatted as a list with a number of elements equal to the number of data slots given as a creation argument to the instance.

```
import datanetwork.*;
DNConnection dn;  // DNConnection instance
DNNode node;      // DNNode instance

void setup() {
  dn = new DNConnection(this, "192.168.0.104",
         dn.getServerPort("192.168.0.104"),
         6009, "p5Client");
  node = new DNNode(2000, 5, 0, "p5Node");
}
void stop() {
  dn.unsubscribeAll();
  dn.removeAll();
  dn.close();
}
void keyPressed() {
    if(key == 'r') dn.register();
    else if(key == 'q') dn.queryAll();
    else if(key == 'f') dn.subscribeNode(401);
    else if(key == 'd') dn.setData(node,
      new float[] { 4.0, 2.0, 1.0, 2.3, 4.4 } );
}
// receive and print data:
void dnEvent(String addr, float[] args) {
    print("Float: " + addr);
    for(int i = 0;i < args.length;i++)
        print(" "+args[i]);
    println();
}
```

Figure 6: An example of using the Processing client.

3.7 Processing and Java

The Processing client is based on the JavaOSC library[18], and is based around two classes: **DNConnection**, dealing with the communication to and from the host, and **DNNode**, dealing with the properties of a DataNode. As the implementation is basically a Java class, the client can not only be used for Processing, but also for any Java program.

An example of use of this library in a program is shown in figure 6.

3.8 C++ library

The C++ library is based upon *liblo*[19] for handling OSC communication and consists of a set of classes to deal with the various components of the DataNetwork:

DataNetwork base class for the network.
DataNode base class for a data node.
DataSlot base class for a data slot.
DataNetworkOSC implements the OSC interface.
OSCServer C++ class to implement an OSC Server (taken from *swonder*[20] and slightly expanded)

[18] http://www.illposed.com/software/javaosc.html
[19] http://liblo.sourceforge.net
[20] http://swonder.sourceforge.net

```
   DataNode * node;
   DataNetwork * dn;

// create a data network:
   dn = new DataNetwork();
// create an osc client interface for it:
// arguments (host IP, client udp port, client name)
   dn->createOSC( ``127.0.0.1'', 7000, ``libdn'' );
// register with the host:
   dn->registerMe();
// query all there is to know about the network
   dn->query();

// subscribe to a node:
   dn->subscribeNode( 5, true );

// create a node:
   dn->createNode( 4, "world", 5, 0, true );
// label one of its slots:
   dn->labelSlot( 4, 2, "hithere", true );

float dummydata[] =  {0.1, 0.3, 0.4, 0.5, 0.6};
// get a reference to the node:
   node = dn->getNode( 4 );
// set data to the node:
   node->setData( 5, dummydata);
// send the data to the network:
   node->send( true );
```

Figure 7: An example of using the C++ library.

An example of use of this library in a program is shown in figure 7.

4 SenseStage Workshop

The first SenseStage workshop[21] was held in May 2009 at Concordia University, as a test case for using many sensor nodes in one space, as well as having a number of artists, unfamiliar with the specifics of the technology and coming from diverse artistic and technical backgrounds, use the DataNetwork simultaneously. Participants were able to employ all available data in various projects, which were developed over the course of one week. While this workshop served as a test case for evaluating the use of our hardware and software, we were also interested in how participants would make artistic use of the potentials of the system.

The workshop resulted in five group projects, with groups consisting of 3 to 5 collaborators, using light, sound, animation and video as output media. The groups were able to go very quickly from concept to experimenting with various sensor modalities and using the data to drive the output media. Encouragingly, the participants seemed to be primarily concerned with "what to do with the data" rather than "how to get the data". The results of this workshop also highlighted the need for more sophisticated ways of dealing with sensor data, for combining, conditioning, and processing of multiple data streams.

Several more SenseStage workshops are planned for 2010 and 2011 as a means to familiarize people with the SenseStage infrastructure, as well as further explore issues in mapping and using large numbers of datastreams.

[21] http://sensestage.hexagram.ca/workshop/

5 Usage cases

In this section, we will discuss several projects in which the SenseStage technology has been used. All of these projects have been, or are being shown at international festivals and multiple venues, thus fullfilling the criterium that the technology has to be useable in a real world, professional artistic environment.

5.1 Dance: Schwelle and Chronotopia

The dance performance Schwelle [Baalman et al., 2007] had been developed before the SenseStage project began and has informed many of the design decisions made during the SenseStage project, so we are currently discussing how the infrastructure can be adapted and improved to use our new technology for future performances. The performance involves 3 accelerometers on the body (originally wireless based upon a Create USB interface with MicroChip RF chips), 1 accelerometer in an object (originally a WiiMote), and 3 light sensing boards (originally wired Create USB interfaces) placed at various places in the room. Furthermore, there is activation of custom lights and motors in one part of the stage. Using the SenseStage MiniBees, we will be able to use now 3 separate sensing boards on the body, saving us problems of wiring along the body; instead of using the WiiMote for the accelerometer in the object, we can now use a SenseStage MiniBee, which we can wake up from a sleep mode, once we need the sensing in the box; this will save us various problems of having to make the WiiMote set up a BlueTooth connection by pushing buttons, while it is packed inside a box. For the sensing boards inside the room, we will also be spared of the wiring. Where we were previously using a custom OSC-namespace for this piece, we can now use the DataNetwork clients instead, and gain much more robustness with regard to reconnecting; also it will be much faster to add new data to exchange between the interactive light controls and the sound control, should we feel the need to do so.

Figure 8: A still from the dance performance Chronotopia with the Attakkalari Centre for Movement.

In Chronotopia, a dance performance by the Bangalore (India) based Attakkalari Centre for Movement, and in collaboration with visual artist Chris Ziegler, we used the wireless technology for controlling a matrix of 6 by 6 cold cathode fluorescent lights (CCFL), and 3 handheld CCFL lights. Since the power required for the light matrix is quite high, it cannot be battery-powered, but the use of wireless technology freed us from running cable between the light matrix and the computer controlling it. Given the short setup time in theaters (usually just one day), and especially in technically challenging environments as India, this was a considerable advantage. For the 3 handheld objects, wireless control was critical, as the objects are carried across the stage by the performers during the show as part of the dramaturgy of the piece. Within the light control setup itself, the DataNetwork was used extensively to exchange data between different portions of the setup, such as the motion tracking data (from a camera looking down at the stage), and pitch and beat tracking data extracted in real-time from the soundtrack. We also exchanged data between the light control and the interactive video, both for synchronisation of cues with the soundtrack (using frametime of the playback, published as data on the network), and for connecting the intensity of the lights to the video image (the light control was publishing the maximum output value of all the lights in the matrix onto the network, which was used to control the brightness of the video image).

5.2 Environmental: MARIN and Arctic Perspective

In two artistic projects dealing with environmental data,
MARIN[22] and Arctic Perspective Initiative[23], the SenseStage MiniBees were used to gather environmental data, such as temperature, humidity, light and air quality, as well as 3-axis acceleration. The DataNetwork was used to access the data for real-time use, and to gather and log all the data to file for artists to use at a later time for visualisation and sonification.

From an expedition to Nunavut in Northern Canada in Summer 2009 as part of the Arctic Perspective Initiative project we learned that the range of the XBee and XBeePros is very much dependent on the environmental conditions. In the outside conditions there, the achieved range of transmission was only a few hundred meters, only about a fifth of the range specified on the datasheet of the XBeePro. At ranges larger than about 50 m. they are extremely affected by blockage, e.g. from the body. In indoor situations, the radio waves will be reflected by objects and walls and such blockage may be mitigated.

Additionally, the batteries lost charge faster in colder conditions (about 6°C) resulting in shorter battery life.

Another issue was that it was difficult to power the host computer, receiving the data from the MiniBees, using solar power, because of foggy and cloudy days. For this reason a kind of datalogger approach for the MiniBees, which stores data locally and sends it to a host when the host is online, could be useful for this kind of application.

Other challenges include finding waterproof housing to protect the electronics from extremely wet weather conditions on sea.

5.3 Installations: JND/Semblance

JND/Semblance is an interactive installation that explores the phenomenon of cross modal

[22] "M.A.R.I.N. (Media Art Research Interdisciplenary Network) is a networked residency and research initiative, integrating artistic and scientific research on ecology of the marine and cultural ecosystems." (from `http://marin.cc/`).

[23] "The Arctic Perspective Initiative (API) is a nonprofit, international group of individuals and organizations whose goal is to promote the creation of open authoring, communications and dissemination infrastructures for the circumpolar region." (from `http://www.arcticperspective.org`).

perception — the ways in which one sense impression affects our perception of another sense. The installation comprises a modular, portable environment, which is outfitted with devices that produce subtle levels of tactile, auditory, visual and olfactory feedback for the visitors, including a floor of vibrotactile actuators that participants lie on, peripheral levels of light, scent and audio sources, which generate frequencies on the thresholds of seeing, hearing and smelling.

In JND/Semblance the SenseStage MiniBees are used for gathering floor pressure sensing data. The SenseWorld DataNetwork is used to gather the raw sensor data, to extract features from it, and to establish flexible mappings to light, sound, and vibration on a platform on which the visitor is lying down.

6 Conclusions

We have presented SenseStage, an integrated hardware and software infrastructure for wireless mesh-networked sensing, actuating, data sharing and composition within interactive media contexts. The infrastructure is unique as it integrates hardware and software, and makes sensor and other data easily available for all collaborators in a heterogeneous media project, within each collaborator's preferred software environment.

We are currently revising the hardware and firmware design, including options to configure and program the boards wirelessly. We plan to have the board available for sale in the second half of 2010. Our future research will focus on techniques for composing and creating with the many streams of realtime data available from such a dense network of sensors.

7 Acknowledgements

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.

Thanks to Matt Biedermann for his feedback on the use of the SenseStage MiniBees in the Arctic Perspective Initiative project.

Thanks to Elio Bidinost, as well as all the SenseStage Workshops participants, for their input and feedback.

7.1 Download

The SenseWorld DataNetwork is available from `http://sensestage.hexagram.ca`. It is released as open source software under the GNU/General Public License.

References

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

M. Beigl, C. Decker, A. Krohn, T. Riedel, and T. Zimmer. 2005. uParts: Low cost sensor networks at scale. In *Ubicomp 2005*.

Sher Doruff. 2005. 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.

Joseph Malloch, Stephen Sinclair, and Marcelo M. Wanderley. 2008. 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.

L. Nachman, R. Kling, R. Adler, J. Huang, and V. Hummel. 2005. The intel mote platform: a bluetooth-based sensor network for industrial monitoring. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN 2005) (Los Angeles, CA, April 2005)*.

Chulsung Park and Pai H. Chou. Eco: Ultra-wearable and expandable wireless sensor platform. In *Third International Workshop on Body Sensor Networks (BSN 2006)*.

J. Ryan and Christopher L. Salter. 2003. Tgarden: Wearable instruments and augmented physicality. In *Proceedings of the 2003 Conference on New Instruments for Musical Expression (NIME-03), Montreal, CA*.

Hans-Christoph Steiner. 2009. Firmata: Towards making microcontrollers act like extensions of the computer. In *Proceedings of NIME 2009 New Interfaces for Musical Expression, Pittsburgh, PA, USA*.