Tom Kobialka
tkob@cs.mu.OZ..AU

June 2008

SensorWeb 2.0 Cookbook

1. Introduction

The purpose of this document is to provide some guidance to the developer / engineer in using SensorWeb 2.0 Software (some places may also be used as NOSA). SensorWeb 2.0 Software is currently in Alpha release stage, some bugs do exist.

Due to time constraints SensorWeb 2.0 Software has not undergone proper deployment testing. Testing has been limited to a handful of programmers.

For the purpose of this document we assume you are using Eclipse as your primary IDE. You should be running Windows XP or some similar variant.

SensorWeb 2.0 / NOSA SVN repository:

svn+ssh://tkob@mundula.cs.mu.oz.au/home/projects/nosa/NOSA

2. Documentation Overview

Prior to running SensorWeb 2.0 Software Users should familiarize themselves with the publications listed at the end of this document.

The following documents have been provided with this release.

a. **NOSA_Imote2IninstallationGuide.doc**: Most recent Installation and Configuration file for NOSA. Includes installation instructions for all required software and sensors including *Imote2, MicaZ, Mica2*.
b. **OSWA-core services.pdf**: Original student report on core services of NOSA written by Xingchen Chu. Good source of developer information including class diagrams. These class diagrams are slightly out of date, however for the most part they should still be very useful.
c. **X20.doc**: Information on the X20.jar program developed by Jehanzeb Khan. X20.jar is used in the SCS and by some clients. It is a parsing application for XML
d. **The Cache Mechanism.doc**: Student report on a cache mechanism developed for the SCS. Contains very good explanation of how caching works, including design documents. *NOTE: Caching capabilities are in the SCS baseline however they have been temporarily disabled due to testing time constraints*.

e. The folder **Documentation/Background_reading** contains useful documentation from third party sources which during the course of development has been useful. If you come across an installation or configuration problem with any third party package this should be your first point of reference.
f. **SensorWeb2-0_Release_Notes.doc**: release notes.

3. Installation and Configuration

To Install NOSA and associated software including
- Cygwin
- TinyOS
- Tomcat
- Globus (Web Services core)
- Postgres

Please follow the instructions provided in *NOSA_Imote2InstallationGuide.doc*

During the course of development *c:/workspace/NOSA* was used as the Eclipse project directory. If you use a different directory you will need to update PATH values in the build scripts listed below.

1. Configuration files:
   a. **NOSA/application.properties**
      Primary configuration file for the Web Services. Located in NOSA/application.properties it is automatically copied to the tomcat directory *Tomcat5.5/common/classes* during build time. In Tomcat various JAR files call upon this file to retrieve information on what classes to execute.

   b. NOSA/buildservice.properties
      Primary configuration file for build scripts. Includes path values for the NOSA project root directory, Tomcat and Globus ws-core.

   c. NOSA/buildservice.xml
      Primary build file. You will need to edit this file in order to build the various services. Generally it is recommended that you build the value nearest to the bottom first, i.e X20, and then work your way up, SensorPlanningService, WebNotificationService, SensorCollectionService, and finally SensorRepositoryService. In oder to build a service, uncomment the service you are interested in, ensure all remaining service names with **<target name ="all"** are commented. Then providing your eclipse has been properly configured to work with Globus and Tomcat click on the "Run build and Deploy" button (see **NOSA_Imote2ININstallationGuide.doc**). Once it has been built click on the "Tomcat" button to launch Tomcat with the service. Then you can use one of the available clients to connect to the service.
   d. NOSA/Cache.conf

Cache configuration file, please refer to the cache documentation for a full description.

e. NOSA/x2o.properties
Configurations file for the X20 library. Refer to the X20 documentation for a full description.

4. Clients:

Once you have configured and installed the environment and the services have been built and are running on the Tomcat containerm, the following are clients which can be used to connect to the Web Services:

1. org.sensorweb.demo.sps.SCSLocatorForm – Connects to the SPS, retives a list of SCS instances which can then be queried by the user. For initializing more than one connection to the SCS (getObservation()) at any one time. Can connect to all sensor network types (techfest, db,nictor,imote). Connection input values are highly configurable. Good visualization.
2. org.sensorweb.demo.DemoFrame – Connects to the SCS (getObservation()). The most stable, can only initialize one connection to the SCS at any one time. Connects to all sensor network types (techfest, db,nictor,imote). Connection input values are highly configurable. No visualization.
3. org.sensorweb.demo.techfest.Talker – Input values are hard coded. Starts two simultaneous connections to the SPS, which connects to the SCS (getObservation()). Connects to nictor and techfest sensor network types. Once a connection has been established the SPS parses resulting observational data. If values from the "Crossbow" (Techfest) light sensors drop below some threshold, the SPS ceases reading values from the NICTOR sensors until they increase. Demonstrates possible scheduling relationship between SPS and SCS. Visualization of observational data on a chart.
4. org.sensorweb.demo.summer.Monitor – Connects to SCS, starts one or more simultaneous connects to the SCS. SCS address value is hard coded, and may need to be changed if it is not localhost. Good for testing other methods besides getObservation e.g. sensorDescription() and platformDescription(). No Visualisation. Only connects to "techfest" sensor network types. Slightly out of date.
5. org.sensorweb.demo.DemoSPSWNSCombo – No GUI interface. Used for testing the WNS and SPS.

Client Setup

Before running any clients, ensure that in eclipse when you right click on the class name and select "Run", go into the run configuration for the class name, ensure that under the "Arguments" tab on the right hand side the following arguments are included:

Program Arguments:

-cp c:\workspace\ws-core-4.0.2\lib\bootstrap.jar -DGLOBUS_LOCATION=c:\workspace\ws-core-4.0.2 org.globus.bootstrap.Bootstrap

VM arguments:

-DGLOBUS_LOCATION=c:\workspace\ws-core-4.0.2

Replace C:\workspace\ws-core-4.0.2 with the location of your Globus WS-Core library folder.

Under the "Classpath" tab add an external folder ("Advanced") and include the location of your Globus ws-core library.

Back in your Eclipse package explorer, select "Run As" to execute the client.

The two primary clients are described below. Similar descriptions can be applied to all other clients.

1. **org.sensorweb.demo.sps.SCSLocatorForm**

   Prior to running SCSLocatorForm ensure that a Tomcat instance is running. This client connects to the SPS service listed in the "SPS url" textbox. It queries the `getServiceInformationImpl() method and r`etrieves a list active SCS instances. SCS instances are read by org.sensorweb.service.globus.sps.impl.SensorPlanningService.java from NOSA/scs-service.xml.

   Num of connections:
   The "Num of Connections" value can be increased from 1 to 32 (tested). "Num of Connections" controls the number of simultaneous connections made by the client to the SCS (from the SCS Locations table).

   "Property", "Operator", "Value" and "NetworkType" are compulsory values for each connection. Certain "Network Types" require can only take certain values, these are listed in Table 1.

NetworkType	Property	Value	Frequency (ms)	Duration (minutes)	SensorType
Techfest	light,temp,sound	> 0	100, 1000	>0	Mica2, Micaz
Imote	Light,temp,volt	>0	100, 1000,	> 0	null

			10000		
Nictor	Light	>0	Null	>0	null
db	Light,temp,sound	>0	Null	>0	null

Table 1: Valid input types for "Num of Connections" fields.

"*Network Types*" correspond to the types (hardware vendor / modle) of sensors connected to the SCS. Configuration values for each network type are located in NOSA/nosa.xml.

- "**Techfest**" Crossbow MicaZ(MPR2400CA & MTS300CA)and Mica2 (MPR400CB & MTS300CA) platforms and sensors which are running nesC code for TinyOS as provided in the following directories:
  - **SensorCode/ NOSADemo** (Mica2)
  - **SensorCode/NOSADemo_MicaZ** (MicaZ)
- **"Imote"** Crossbow Imote (IPR2400 & ITS400CA) platform and sensors which are running code from
  - **SensorCode/NOSADemo_Bas**
- **"Nictor"** NICTA designed and developed sensors specialized for soil moisture and water monitoring purposes. This requires Microsoft SQL Server 2005 to be setup and running in order to interface with the NICTOR's.
- **"db"** TinyDB simulation of sensor network. TinyViz must be up and running.

"*Sensor Type*" is only applicable to techfest. Select the appropriate sensor type (MicaZ, Mica2) to correspond with your sensors.

"*Frequency*" is an optional value, valid frequency values are listed in Table 1. Frequency denotes the time (in ms) between which observational values are sampled by sensors and forwarded to the base station.

"*Duration*" is an optional value. This will start a connection to the SCS which will persist for the duration period (in minutes). Updates are automatically returned to the SCS after a successful observation query completes on the selected sensor network. Several updates may be returned within the duration period, this depends on the "Frequency" value.

2. org.sensorweb.demo.DemoFrame

"PropertyName", "Operator" and "Value" are compulsory. ObservDuration corresponds to the duration of a query, it is optional, likewise is "Frequency". Check the org.sensorweb.demo.SCSLocatorForm descriptions for a full outline of these.

“SensorDescrpt” and “Pltfrm Descrpt” are retrieving the sensor and platform descriptions for (hardcoded) values from the SCS.

Ignore the “Progress” bar, “HistoricObservation” button, “Streaming Data Based Mode” tab and “TinyDB Application” pull down box.

2 Setting up the Sensors

SerialForwarder is an application which runs in the background and acts as a communication bridge between the SCS and the sensors. It is a requirement to have an instance of SerialForwarder up and running if you want to communicate with any crossbow sensor running TinyOS.

Please only use the SerialForwarder version provided in the NOSA/net directory. It contains a bug fix that allows more than one SerialForwarder instance to execute at anyone time.

For MicaZ or Mica2 sensors, be sure to 'unset TOS_PLATFORMS', then
In a cygwin terminal window go to /NOSA/net andexecute:

For MICA2: **java net.tinyos.sf.SerialForwarder -comm serial@COMX:mica2 -port 9001**

For MICAZ: **java net.tinyos.sf.SerialForwarder -comm serial@COMX:57600 -port 9002**

Where COMX is the serial port which the Base Station is connected to.Active serial ports can be discovered in your Device Manager.

If Imote2 or Mica2 sensors are connected be sure to set TOS_PLATFORMS
I use:

export TOS_PLATFORMS=c:/workspace/.platform.properties

'cat .platform.properties' should look like this:

mica=avrmote,2,19200
micaz=micaz,3,57600
mica2=avrmote,2,57600
imote2=micaz,3,115200

The execute:

IMOTE2: **java net.tinyos.sf.SerialForwarder -comm serial@COMX:imote2 -port 9003**

If you experience MSG_Length errors in SerialForwarder, typically you TOS_PLATFORMS variable has not been set correctly.

2.2 Techfest (MicaZ & Mica2)

- TOSBase is nesC code which is provided with the TinyOS installation of Cygwin. TOSBase should be installed on the basestation node in order to communicate with the remainder of the sensors. TOSBase is located in $TOSROOT/apps/

  Open $TOSROOT/tos/system/tos.h
  find: uint16_t TOS_LOCAL_ADDRESS = 1;
  Change the value to be 0 for the basestation, it must be unique for each mote.

  Connect the docking station to the serial port and power source. Physically place the Mote into the base docking station (remove the sensorboard) Ensure all SerialForwarder instances or other applications which may be holding COM port resources have been exited.

  In $TOSROOT/apps/TOSBase execute:

  > make mica2 install mib510./dev/ttyS0

  Replace mica2 with micaz if you are making for MicaZ sensors. /dev/ttyS0 refers to your serial port in this case it is COM1, /dev/ttyS1 refers to COM2, and so on.
  Your basestation should now be programmed.

- To program the remainder of the motes you will need to copy NOSA/SensorCode/NOSADemo (Mica2) or NOSA/SensorCode/NOSADemo_MicaZ to $TOSROOT/apps.

  Open $TOSROOT/tos/system/tos.h
  find: uint16_t TOS_LOCAL_ADDRESS = 0;
  Change the value to be 1 for the first node, it must be unique for each mote.

  Remove the programmed basestation Mote from the base docking station (remove the sensorboard). And place the mote you want to program in it's place.

  In the relevant $TOSROOT/apps directory execute

> make mica2 install mib510./dev/ttyS0

Once you have programmed all your motes, be sure to replace the mote programmed with TOSBase into the docking station so that you can communicate with the other motes.

Known Bugs

- Error with temperature sensor (see imote2 known bugs below).

2.3 Imote2

You will need to select one of the imote2 motes to act as the basestation. Connect this mote via USB to the machine. Edit tos.h and set TOS_LOCAL to be 0.Go to, $TOSROOT/apps/TOSBase and execute:

>make imote2 debug

For the remainder of the motes, code for the Imote2 Sensors is located in:

NOSA/SensorCode/NOSADemo_Bas

The SCS can only read from sensors which are running this code. If you want to run different code on your sensors you must do as follows

Once you have created your *.nc you will need to use **mig java** to generate your java interfaces. Have a look at NOSA/SensorCode/NOSADemo_Bas/jMakefile to see how this can be done. These interfaces will then need to be called by a connector class, such as *org.sensorweb.core.scs.tinyos.imote.ImoteConnector* along with the expected input values.

Note: http://ics.yeditepe.edu.tr/tnl/html/LOCAL/files/docs/tos-source-tree/ is an excellent source for TinyOS API information

In order to install the code copy NOSA/SensorCode/NOSADemo_Bas to $TOSROOT/apps directory,

- Edit $**TOSROOT/tos/system/tos.h**
  Replace x
  uint16_t TOS_LOCAL_ADDRESS = x;
  With an integer, each sensor should be identified by a unique integer.

> cd $TOSROOT/apps/NOSADemo_Bas
> make imote2 debug

- plug your sensor into the USB port

> USBLoaderHost.exe –p build/imote2/main.bin.out

This will upload the code to your sensors.
To run the code launch the SCS using Tomcat, launch the serialForwarder instance located in **NOSA/net/** select the DemoFrame or SCSLocatorForm clients, construct an input string and select "imote" as the networkType, and launch the query.

Known Bugs:

1. Currently there is a bug in the NOSADemo_Bas code. If "temp" is selected as the input sensor type the imote2 sensors will continue sensing even after the SCS has sent a **packet.set_action(SENSING_OFF)** request. For this reason readings from the Temperature sensor should be avoided. Observational values from light and voltage sensors should be used instead.
2. In org.sensorweb.core.scs.tinyos.imote.ImoteConnector, process(…) when set_action(interval) is called the value passed across to *NOSADemo_Bas.nc* (running on the sensors) should be in milliseconds.

NOTE: http://tech.groups.yahoo.com/group/intel-mote2-community/ is an excellent source for Imote2 help.

2.4DB (TinyDB)

Instructions for TinyDB are provided in **NOSA_Imote2ININstallationGuide.doc**

2.5 NICTOR

Terracept Terraview setup file located in SensorCode/Nictor/TerraviewTest.nwk

3 Sensor Collection Service

The SCS is by far the most complete service out of all the services.

A Postgres database is a dependency for the methods sensorDescription() and platformDescrption(). The SQL for the database is located in the NOSA/sql folder.
The relevant SQL is contained in the following files:

- Sensorml_create_tables.sql
- Sensorml_data.sql
- Sensorml_drop_tables.sql

The following methods are implemented in the SCS:

1. getObservation()

The most amount of work has gone into this method. It is used to communicate with the sensor network and retrieve observational results. It can handle simultaneous connections to more than one sensor network at any one time.

In addition the SCS has the capacity to temporarily cache results observational results produced by the sensor network in order to improve performance. This code has been included in the baseline, but has been temporarily disabled. In order to enable the cahce, the following files need to be edited:

- *org.sensorweb.core.scs.DBProxy.java* : Cache implementation using TinyDB/TinyViz produced simulation data
- org.sensorweb.core.demo.Techfest.TechSensorProxy.java: Cache of data produced by MicaZ/ Mica2 sensors.

The cache is only applicable to data produced by TinyViz/TinyDB or Crossbow MicaZ/Mica2 sensors.

A timeout value has been implement in TechConnector.java and ImoteConnector.java such that if no response is received from the sensor network after a minute a null value will be returned. This prevents the SCS from hanging if the sensor network is down.

2. getObservationDuration()
   This is a duplicate of getObservation(). Observational queries with a duration value which require the SCS to persist connect to this method.

3. sensorDescripition()
   Retrieves a description, form a Postgres Database, of a sensor type. The input string should be in the format of "Sensor_1".

4. platformDescription()
   As for sensorDescription(), however it retrieves a platform description instead. Input string should be in format "Platform_1".

Have a look at the client org.sensorweb.demo.summer.Monitor, for examples on how to invoke sensorDescription() and PlatfromDescription().

Known Bugs

a. When a client constructs a request to the SCS it will typically contain at least three items, i.e **light** (sensor type) **>** (operator) **23** (value). Currently the value is largely ignored. That is to say, for a observational query all light values will be returned, regardless of if they are over 23 or not.

Although this is not really a bug, it is something that has been fixed in the cache code but not yet propagated through to the baseline due to time constraints.

b. Currently when observational data is read from a sensor network only a limited amount of values (10) are read from the network as a *whole* (regardless of how many nodes there are). This means that nodes with a greater signal strength will send more packets back to the base station than those with a lower strength. This leads in an uneven distribution of observational values read from the sensor network. To solve this issue the amount of observational values *per node* need to be limited. An example would be to set the limit to 10 observational values per node.
c. The timeout value only returns null. However this should be changed so that a more detailed message of exactly what went wrong is returned. This has been partially implemented in ImoteConnector.java, with the holder being set to "timeout". However this change has not fully been replicated therough to the XMLBuilder which converts the ObservationCollection object into XML.
d. The Encoding of the XML is inefficient. XML is converted from w3c to jdom and then into a object (i.e. ObservationCollection). This needs to be redone, so that either jdom or w3c is used, and objects are eliminated. In particular org.sensorwebi.model.* needs to be removed and redone. This is applicable to all services.
e. With Imote2 the SerialForwarder often reads packets which cause the error "packet too long". In the SCS is displays the error "Dropping packet with bad group". Time constraints have prevented further investigations into this problem.
f. When there are 2 or more sensors connected at any one time and more than one concurrent connection is being executed (with durations), there is no guarantee that the SCS will pay equal attention (read the same amount of data) from both of the sensor networks. This is largely left up to chance, on network might be queried 4 times, while the other might wait for the resource to become available and will consequently only read 2 values before its duration has expired. An accounting scheme must be implemented, such that each connection is guaranteed the same amount of observation data and time from each sensor network.
g. Sometimes, even though a close() call has been made, the sensors keep on sensing. Happens when there are several simultaneous connections to more than one sensor network.

4 Sensor Repository Service

The SRS is a repository service for storing historic observational data produced by the SCS. It is essentially an interface through to a Postgres Database.

Code for the SRS is located in

- org.sensorweb.service.srs
- org.sensorweb.service.srs.impl

The SRS implements the method:

- saveObservation(Element request)

Element request must be a ObservationCollection XML type.

Postgres must be installed and configured in order for the SRS to function. SQL data for the SRS is located in

- NOSA/sql/SRS_create_tables.sql
- NOSA/sql/SRS_drop_tables.sql

5 Web Notification Service

The WNS is a skeleton service with a few interfaces implemented. Have a look at the OGC WNS document to understand potential development paths.
Postgres must be running in order for this service to function, SWL is located in:

- NOSA/sql/WNS_create_tables.sql
- NOSA/sql/WNS_drop_tables.sql

6 Sensor Planning Service

The Sensor Planning Service is largely hardcoded with values particular to various demonstrations. It needs to be modularized and generalized, the OGC SPS specification document provides more information on the SPS.

1. SubmitRequest(): This routine has largely been hardcoded to work with org.sensorweb.demo.techfest.Talker.java, it demonstrates a potential use of the SPS in scheduling connections to the SCS and acting as a intermediary for processing observational results.
2. getSCSLocations(): This method returns a list of SCS addresses which have been registered with it. org.sensorweb.demo.sps.SCSLocatorForm demonstrates the potential use for this method.

7 Debugging

Network traffic can be sniffed using TcpTunnelGui

cd /cygdrive/c/workspace/soap-bin/soap-2_3_1/lib
java -cp soap.jar org.apache.soap.util.net.TcpTunnelGui 8081 localhost 8080

In order to use this you will need to change the Service addresses to connect to port 8081, instead of 8080. This information is located in NOSA/nosa.xml and the application.properties file located in your tomcat directory. For me this is: C:\Program Files\Apache Software Foundation\Tomcat 5.5\common\classes\application.properties

Tomcat

change in your JDK lib (/cygdrive/c/ProgramFiles/Java/jdk1.5.0_02) /jre/lib/logging.properties cosole.logging.level from FINE to FINEST. This increases the granularity of Java classes printed when running Tomcat.

Publications

- Xingchen Chu, Tom Kobialka, Bohdan Durnota, and Rajkumar Buyya, Open Sensor Web Architecture: Core Services,, Proceedings of the 4th International Conference on Intel ligent Sensing and Information Processing (ICISIP 2006,IEEE Press, Piscataway, New Jersey, USA, ISBN 1-4244- 0611-0, 98-103pp), Dec. 15-18, 2006, Bangalore, India.

- Tom Kobialka, Rajkumar Buyya, Christopher Leckie, and Rao Kotagiri, A SensorWeb Middleware with Stateful Services for Heterogeneous Sensor Networks, Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2007, IEEE Press, Piscataway, New Jersey, USA), Dec. 3-6, 2007, Melbourne, Australia.

- Xingchen Chu and Rajkumar Buyya, Service Oriented Sensor Web, Sensor Network and Configuration: Fundamentals, Techniques, Platforms, and Experiments, N. P. Mahalik (ed), Springer-Verlag, Germany, 2007.