# Update of the SGX2 Project – December, 2011

## Goals

Design a clean API for the SGX Project.

## The Source Location

The source code for the project can be found at the location

<https://sensorcloud.uits.indiana.edu/svn/Experimental/sgx2>

Use svn to check out the code.

svn co <https://sensorcloud.uits.indiana.edu/svn/Experimental/sgx2> sgx2

## How to build

Apache maven is used as the build tool for the project. To build the project, execute the following command at the root directory of the project.

mvn clean install

This will build the sgx project and a zip distribution file will be created in the module/distribution/target directory.

### IDE’s

It is possible to generate the project files for your favorite IDE using Maven.

#### Eclipse

mvn eclipse:eclipse

#### IntelliJ Idea

mvn idea:idea

These commands will generate the corresponding project files for the given IDE’s.

## How to run

### Starting SGX Server

Extract the zip distribution and go to the bin directory. Execute the command scserver.sh/bin.

This will bring up the SGX Server.

### Starting the Apache ActiveMQ

It is required to start an Apache ActiveMQ instance which is used as the message broker. Download Apache ActiveMQ, extract and go to bin directory. Execute the command

./activemq start

### Running the Sample Chat Sensor and Client

1. Start Apache ActiveMQ server and SGX server as described in the previous sections.
2. Build the Java IDE project of SGX for IntelliJ idea or Eclipse using maven.
3. Then go to the Sensors module and run the ChatSensor class.
4. Next go to the Clients module and run the ChatClient class.
5. Now type a word in the ChatSensor command line console. This message will be sent to the ChatClient as a data message and will be displayed, in its command line window.
6. Now type a word in the ChatClient in the command line console. This message will be sent to the ChatSensor as a control message and will be displayed, in its command line window.

## Overview

The objective of the Sensor Cloud Project is to provide a general-purpose messaging system for sensor data called the *Sensor Grid Server*, and provide a robust *Application API* for developing new sensors and client applications. The key design objective of the Sensor Grid API is to create a simple integration interface for any third party application client or sensor to the Sensor Grid Server. This objective is accomplished by implementing the *publish/subscribe* design pattern, which allows for loosely coupled, reliable, scalable communication between distributed applications or systems.

Sensor Cloud

Receive, Subscribe, Notify

Client Application Enterprise App

Client Application Desktop Client

Client Application Web Client

Publish

Publish

Notify

Notify

Notify

Publish

## Overall Design

The SGX is comprised of four parts. They are

1. SGX Middleware
2. Message Broker
3. Sensors
4. Clients

The SGX Middleware acts as a central hub for managing the communications between Sensors and Clients. Sensors and Clients exchange data through a message broker. A Sensor is an entity, which can produce data and accepts control messages from the Clients. A Client is an entity, which can subscribe to Sensor messages and control the behavior of Sensors.

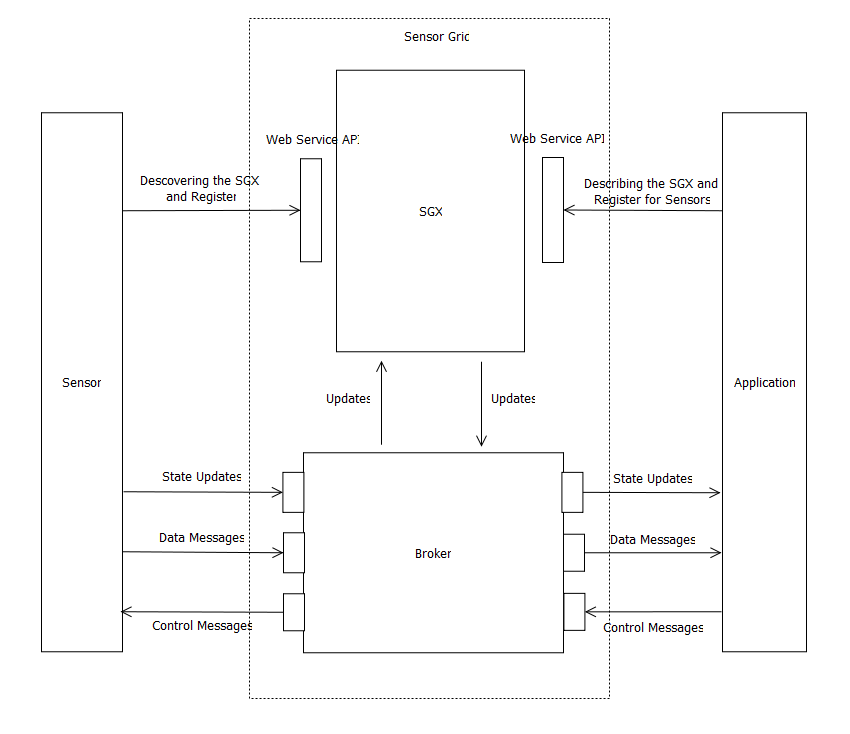


Figure - Overall Design

SGX Middleware component is managing the Message Broker, Sensors and Clients.

SGX exposes a Web Services API for clients and sensors. Sensors use this API for getting information about the SGX as well as register itself for the SGX. The information received includes communication protocols and ports.

Once a sensor successfully registers itself with the SGX, it can send data messages and receive control messages through the Message Broker. A separate message broker handles actual data messages. In the initial stages only text messages are supported and JMS [1] style broker (ActiveMQ) will be used. For the streaming message case, it is required to develop a streaming broker with the capability of message repeating with some routing logic.

The Sensor clients use the Web Services API exposed by the SGX to discover the information about the SGX as well as to register itself to the SGX. The clients can start listening on the data channels. At the same time they can send control messages to the sensors. Receiving the data and sending the control messages are done via the Message Broker.

Clients can register for the global state updates about the SGX, as well as updates about the sensor it is receiving data from.

SGX doesn’t assume any particular data format for the Sensors and Clients. It only defines the data formats for the update messages. The message format is a contract between specific sensors and clients.

## Deployment

A deployment will consist of a broker cluster and a SGX cluster. Since SGX doesn’t involve in actual data message flows, the load imposes on it, is pretty simple. As a result, it can be clustered for high availability.

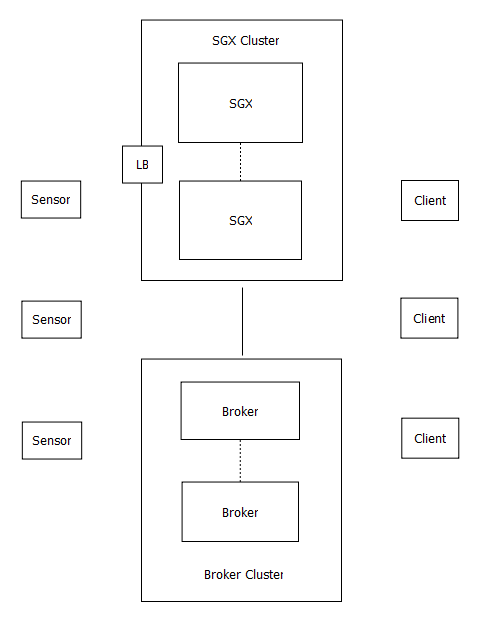


Figure - Deployment

## Design

The design consists of three major modules.

1. SGX Core
2. Sensors
3. Clients

### SGX Core

SGX Core has the entire basic infrastructure for managing the clients, sensor and broker. This module has classes for defining sensors, clients and messages. Also it embeds Eclipse Jetty[2] and Apache Axis2 [3] for hosting the web services.

SGX Server is a part of the SGX core. Startup script is used to start the server. When SGX server starts, it brings up a Jetty Sever and deploys Axis2 Web Services on to this Jetty Server. Once the server starts, it listens for updates send by the clients.

When a Client or Sensor registers to the SGX server, it manages its information. At the same time, it provides information to the entity, which is for discovering the information needed for communication through the broker.

The SGX Configuration is done through XML. The broker-config.xml can be found in the repository/conf directory of the binary distribution which contains this configuration.

### Sensor

This module is a helper for creating sensors. It provides an easy to use API for sensor developers, to hide the underlying messaging for discovering and registering sensors.

A sensor developer should use this module as a dependency and should use the helper classes provided with it.

When a sensor tries to register to the SGX, this module communicates that information to the SGX and retrieves information about the communication channels and populates the sensor as necessary.

### Client

This module is a helper for creating clients. It provides an easy to use API for client developers, to hide the underlying messaging for discovering and registering clients.

A client developer should use this module as a dependency and should use the helper classes provided with it.

## The current progress

In the beginning I was going through the research papers and the existing SGX code base to understand, what are the problems it tries to solve and the design methods that has been employed.

The basic APIs for sensors, messaging and clients are being completed at the moment. End to end scenarios for block messages is also being completed.

Next major implementation will be streaming broker and It will be used to solidify the messaging APIs.

## Future work

### Broker Distribution

Broker distribution is necessary for horizontal scaling of the SGX. Since the broker handles the bulk of the messages, it imposes a heavy load on the broker.

In a static deployment, brokers can be pre-configured and deployed according. If the deployment is dynamic as in a cloud environment, a dynamic provisioning approach should be taken for deploying brokers according to the demand.

### Streaming

It is required to handle streaming messages, like video streams coming from sensors. For this type of messages a raw streaming broker is suitable and Netty [4] project by JBoss provides a library for developing such applications.

### Cloud Provisioning

In a cloud environment, dynamic provisioning can be used for brokers.

### Authentication/Authorization

Support authentication and authorization for clients and sensors with secure communication channels. The authentication and authorization can be based on Apache Shiro [5], which provide user management capabilities.

# Bibliography

x

|  |  |
| --- | --- |
| [1] | JCP. (2011, Nov) JMS. [Online]. <http://www.jcp.org/en/jsr/detail?id=914> |
| [2] | Eclipse. (2011, Nov) Jetty. [Online]. <http://www.eclipse.org/jetty/> |
| [3] | Apache. (2011, Nov) Axis2. [Online]. <http://axis.apache.org/axis2/java/core/> |
| [4] | JBoss. (2011, Nov) Netty. [Online]. <http://www.jboss.org/netty> |
| [5] | Apache. (2011, Nov) Shiro. [Online]. <http://shiro.apache.org/> |
| [6] | Apache. (2011, Nov) ActiveMQ. [Online]. <http://activemq.apache.org/> |

x