**What is the Sensor Cloud Project?**

Contents

[Introduction 2](#_Toc287788173)

[Publish/Subscribe Architecture 2](#_Toc287788174)

[Sensor Cloud Overview 3](#_Toc287788175)

[Sensor Cloud Architecture 5](#_Toc287788176)

[References 9](#_Toc287788177)

# Introduction

The Indiana University Pervasive Technology Institute is developing a Sensor-Centric Middleware System hereafter referred to as the Sensor Cloud.

The objective of the Sensor Cloud Project [[1-3](#_ENREF_1)] is to provide a general-purpose messaging system for sensor data called the *Sensor Grid Server*, and 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 [[4](#_ENREF_4)] which allows for loosely-coupled, reliable, scalable communication between distributed applications or systems.

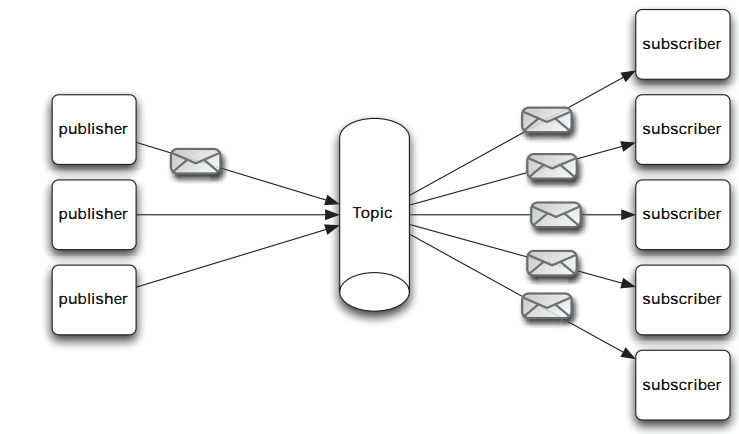
# Publish/Subscribe Architecture

The publish/subscribe (pub/sub) design pattern [[5](#_ENREF_5)] describes a loosely-coupled architecture based message-oriented communication between distributed applications. In such an arrangement applications may fire-and-forget messages to a broker that manages the details of message delivery. This is an especially powerful benefit in heterogeneous environments, allowing clients to be written using different languages and even possibly different wire protocols. The pub/sub provider acts as the middle-man, allowing heterogeneous integration and interaction in an asynchronous (non-blocking) manner.

The pub/sub architecture uses destinations known as *topics*. Publishers address messages to a topic and subscribers register to receive messages from the topic. Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to the content hierarchy. The system takes care of distributing the messages arriving from a topic's multiple publishers to its multiple subscribers. Topics retain messages only as long as it takes to distribute them to current subscribers.  Figure 1 illustrates pub/sub messaging.

Message publication is inherently asynchronous in that no fundamental timing dependency exists between the production and the consumption of a message. Messages can be consumed in either of two ways:

* **Synchronously.** A subscriber or a receiver explicitly fetches the message from the destination by calling the receive method. The receive method can block until a message arrives or can time out if a message does not arrive within a specified time limit.
* **Asynchronously.** A client can register a *message listener* with a consumer. A message listener is similar to an event listener.



*Figure 1 Elements of a Publisher/Subscribe System*

A publish/subscribe system can be conveniently implemented using a Java Messaging Service (JMS) [[6](#_ENREF_6)] compliant Message-Oriented Middleware (MOM) such as NaradaBrokering [[7](#_ENREF_7), [8](#_ENREF_8)], ActiveMQ [[9](#_ENREF_9)], SonicMQ [[10](#_ENREF_10)] etc. to handle message mediation and delivery.

# Sensor Cloud Overview

The Sensor Cloud implements the *publish/subscribe* design pattern to orchestrate communication between sensors and client applications which form an inherently distributed system.

* Sensor Cloud Server creates *Publisher-Subscribe Channels* (Represented as a JMS Topic)
* Sensors acting as publishers create *TopicPublishers* to send messages to a Topic
* Client applications acting as subscribers create *TopicSubscribers* to receive messages on a topic
* Apache ActiveMQ is used as the default underlying MOM and any other JMS style broker can be used as well.

Figure 2 shows a high-level overview of a typical deployment scenario for the Sensor Grid. Sensors are deployed by the Grid Builder into logical domains; the data streams from these sensors are *published* as topics in the sensor grid to which client applications may *subscribe*.

Publish

Sensor Grid

-Subscribe()

-Notify()

-Unsubscribe()

Client Application Enterprise App

Client Application Desktop Client

Client Application Web Client

Publish

Publish

Notify

Notify

Notify

*Figure 2 Schematic of the Sensor Cloud*

Examples of physical devices already implemented include:

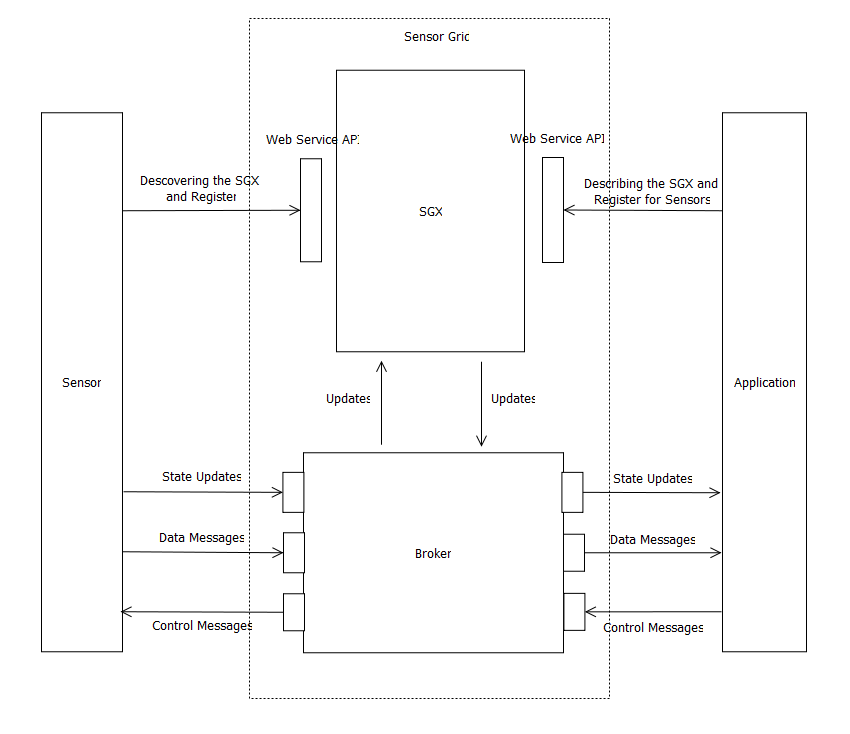
* Web Cameras
* Wii Remotes
* Lego MindStorm NXT Robots
* Bluetooth GPS Devices
* RFID Readers

However Sensors can be made from chat clients, Power Point presentations, web pages virtually anything which produces data in a time-dependent stream can be implemented as a Sensor Grid sensor.

# Sensor Cloud Architecture

The main objective of the Sensor Cloud Project is to design and develop an enabling framework to support easy development, deployment, management, real-time visualization and presentation of collaborative sensor-centric applications. The Sensor Grid framework is based on an event-driven model that utilizes a pub/sub communication paradigm over a distributed message-based transport network.

The Sensor Grid is carefully designed to provide a seamless, user-friendly, scalable and fault-tolerant environment for the development of different applications which utilize information provided by the sensors. Application developers can obtain properties, characteristics and data from the sensor pool through the Sensor Grid API, while the technical difficulties of deploying sensors are abstracted away. At the same time, sensor developers can add new types of sensors and expose their services to application developers through Sensor Grid’s Sensor Service Abstraction Layer (SSAL). ActiveMQ (NB) is the transport-level messaging layer for the Sensor Grid. The overall architecture of the Sensor Grid is shown in Figure 3.

****

*Figure 3 Sensor Grid Components*

**Sensor Grid Server** (SG)

The SG mediates collaboration between sensors, applications and the GB. Primary function of SG is to manage and broker sensor message flows.

* Sensor/SG flow - The SG keeps track of the status of all sensors when they are deployed or disconnected so that all applications using the sensors will be notified of changes. Sensor data normally does not pass through SG.
* Application/SG flow - Applications communicate application API, which in turn communicates with SG internally. Applications can define their own filtering criteria, such as location, sensor id, and type to select which sensors they are interested in. These filters are sent to SG for discovering and linking appropriate sensors logically for that application and forwards messages among the relevant sensors and that application. SG must always check which sensors meet the selected filter criteria and update the list of relevant sensors accordingly. It then sends an update message to application if there are any changes of the relevant sensors.
* Sensors’ properties are defined by the sensors itself. Applications have to obtain this information through SG.
* Application/Sensor flow – The SG provides each application with information of sensors they need according to the filtering criteria. The application then communicates with sensors through the application API for receiving data and sending control messages.

**Application API**

The Sensor Grid aims at supporting a large amount of applications for users and service providers of different industries (e.g. financial, military, logistics, aerospace etc.). The Sensor Grid provides a common interface which allows any kind of application to retrieve information from the sensor pool managed by SCGMMS. The API also provides filtering mechanism which provides application with sensors matching their querying criteria only.

**Sensor**

The definition of sensor is a time-dependent stream of information with a geo-spatial location. A sensor can be a hardware device (e.g. GPS, RFID reader), a composite device (e.g. Robot carrying light, sound and ultrasonic sensor), Web services (e.g. RSS, Web page) or task-oriented Computational Service (e.g. video processing service).

**Sensor Client Program**

A sensor needs a Sensor Client Program (SCP) to connect to the Sensor Grid. The SCP is the bridge for communication between actual sensors and SCGMMS. On the sensor side SCP communicates with the sensor through device-specific components such as device drivers. On the Sensor Grid side SCP communicates with the Sensor Grid through the Sensor Service Abstraction Layer.

# References

1. Geoffrey Fox, Alex Ho, Rui Wang, Edward Chu, and Isaac Kwan, *A Collaborative Sensor Grids Framework*, in *2008 International Symposium on Collaborative Technologies and Systems (CTS 2008)*. May 19-23, 2008. The Hyatt Regency Irvine, Irvine, California, USA. <http://grids.ucs.indiana.edu/ptliupages/publications/CTS08_paper_final.pdf>.

2. Geoffrey Fox, Alex Ho, Eddy Chan, and William Wang, *Measured Characteristics of Distributed Cloud Computing Infrastructure for Message-based Collaboration Applications*, in *International Symposium on Collaborative Technologies and Systems CTS 2009*. May 18-22, 2009, IEEE. The Westin Baltimore Washington International Airport Hotel Baltimore, Maryland, USA. pages. 465-467. <http://grids.ucs.indiana.edu/ptliupages/publications/SensorClouds.pdf>. DOI: 10.1109/cts.2009.5067515.

3. *Community Grids Laboratory Sensor Grid Project* [accessed 2011 March 13]; Available from: https://sites.google.com/site/sensorcloudproject/.

4. P. Th. Eugster, P. A. Felber, R. Guerraoui, and A. Kermarrec, *The Many Faces of Publish/Subscribe.* ACM Computing Surveys, June, 2003. **35**(2): p. 114-131.

5. G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. 2003, Addison-Wesley. p. 207-222.

6. *Java Message Service JMS Version 1.4* [accessed 2011 March 13]; Sun Microsystems (Oracle) Available from: <http://download.oracle.com/javaee/1.4/tutorial/doc/JMS2.html>.

7. Pallickara, S. and G. Fox, *NaradaBrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids*, in *ACM/IFIP/USENIX 2003 International Conference on Middleware*. 2003, Springer-Verlag New York, Inc. Rio de Janeiro, Brazil.

8. NaradaBrokering. *Scalable Publish Subscribe System*. 2010 [accessed 2010 May]; Available from: <http://www.naradabrokering.org/>.

9. Apache. *ActiveMQ open source messaging system*. [accessed 2010 November 27]; Available from: <http://activemq.apache.org/>.

10.  *SonicMQ Robust Enterprise Messaging*. [accessed 2011 March 13]; Progress Software Available from: <http://web.progress.com/en/sonic/sonicmq.html>.