AFRL-RY-WP-TR-2012-0160

ADAPTIVE MULTI-LAYERED SENSING ARCHITECTURES (AMSA)

Task Order 0004: Advanced Technology for Sensor Clouds

Stephen Halwes, Dale Williams, and Gary Whitted
Ball Aerospace & Technologies Corp.

Alex Ho
Anabas, Inc.

Ryan Hartman
Indiana University

MAY 2012
Final Report

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

AFRL-RY-WP-TR-2012-0160 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH THE ASSIGNED DISTRIBUTION STATEMENT.

[illegible]
LISA JONES, Program Manager
Distributed Collaborative Sensor System Tech
Integrated Electronic & Net-Centric Warfare

[illegible]
KENNETH LITTLEJOHN, Chief
Distributed Collaborative Sensor System Tech
Integrated Electronic & Net-Centric Warfare

[illegible]
TODD KASTLE, Division Chief
Integrated Electronic & Net-Centric Warfare
Sensors Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE ***(DD-MM-YY)*** May 2012	**2. REPORT TYPE** Final	**3. DATES COVERED** ***(From - To)*** 05 October 2010 – 06 April 2012

4. TITLE AND SUBTITLE ADAPTIVE MULTI-LAYERED SENSING ARCHITECTURES (AMSA) Task Order 0004: Advanced Technology For Sensor Clouds	**5a. CONTRACT NUMBER** FA8650-09-D-1500-0004
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 62204F
6. AUTHOR(S) Stephen Halwes, Dale Williams, and Gary Whitted (Ball Aerospace & Technologies Corp.) Alex Ho (Anabas, Inc.) Ryan Hartman (Indiana University)	**5d. PROJECT NUMBER** 6095
	5e. TASK NUMBER 22
	5f. WORK UNIT NUMBER 60952404
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Ball Aerospace & Technologies Corp. Systems Engineering Solutions 2875 Presidential Dr., Suite 180 Fairborn, OH 45324-6269 / Anabas, Inc. ---------------------------------- Indiana University	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory Sensors Directorate Wright-Patterson Air Force Base, OH 45433-7320 Air Force Materiel Command United States Air Force	**10. SPONSORING/MONITORING AGENCY ACRONYM(S)** AFRL/RYWC
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-RY-WP-TR-2012-0160

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution authorized to Department of Defense and U.S. DoD contractors only; Administrative or Operational Use; May 2012. Refer requests for this document to AFRL/RYWC, Wright-Patterson Air Force Base, OH 45433-7320.

13. SUPPLEMENTARY NOTES
Report contains color.

14. ABSTRACT
This report details the overall research and development (R&D) activity accomplished on Adaptive Multi-Layered Sensing Architectures (AMSA) Task Order (TO) 4, Advanced Technology for Sensor Clouds. This goal of the research described herein is to conduct research, develop technology and components, and integrate the results for prototyping scalable cloud computing and advanced sensor management services into a multi-layered sensor grid testbed. The research team, composed of Ball Aerospace & Technologies Corporation (BATC), Indiana University (IU) and Anabas research personnel, worked closely with AFRL/RYW personnel to complete the AMSA TO 4, Advanced Technology for Sensor Clouds, research. The IU and Anabas researchers focused primarily on the R&D associated with enhancing the core infrastructure sensor grid middleware to function using cloud computing features, along with other key enhancements. The BATC researchers built the Sensor Grid Testbed functionality on top of the Sensor Grid Middleware and built the application components required for the LVC SIDFOT program. This final report documents the research completed on the AMSA TO 4 effort, along with lessons learned, recommendations, and conclusions.

15. SUBJECT TERMS
adaptive multi-layered sensing architectures (AMSA), trustworthiness, sensor network, narada broker

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT:**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** (Monitor)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Lisa Jones
Unclassified	Unclassified	Unclassified	SAR	174	**19b. TELEPHONE NUMBER** *(Include Area Code)* (937) 528-8018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 Summary

This report covers the results associated with the research and development of concepts, methodologies, tools, and techniques for using sensor clouds for providing operational situational awareness to the 21st century warfighter.

The goal of the research described herein was to conduct research, develop technology and components, and integrate the results for prototyping scalable cloud computing and advanced sensor management services into a Multi-Layered Sensor Grid testbed. In turn, the Sensor Grid testbed provides an environment for conducting advanced trustworthiness related research associated with multi-layered sensor systems operating in urban operation scenarios. Specifically, this research leveraged the efforts of previous sensor grid technology research accomplished under related Air Force Research Laboratory (AFRL) Sensors Directorate (AFRL/RY) research efforts to develop a Sensor Grid Testbed with technologies that will, in turn, be used to support a cross-directorate research project called Live, Virtual, and Constructive (LVC) Sensors Integration for Data Fusion in Operations and Training (SIDFOT).

The research and development (R&D) accomplished on this Adaptive Multi-Layered Sensing Architectures (AMSA) Task Order (TO) 4 encompassed four major research thrusts: (1) enhancing the Core Infrastructure Sensor Grid Middleware; (2) developing an enhanced sensor grid application; (3) researching and implementing trustworthiness algorithms; and (4) prototyping, developing, integrating and demonstrating the resulting technologies. Additional detail on the research associated with these individual projects is provided in the following sections.

The research team, composed of Ball Aerospace & Technologies Corporation (BATC), Indiana University (IU), and Anabas research personnel, worked closely with AFRL/RYW personnel to complete the AMSA TO 4, Advanced Technology for Sensor Clouds, research. The IU and Anabas researchers focused primarily on the R&D associated with enhancing the Core Infrastructure Sensor Grid Middleware to function using cloud computing features, along with other key enhancements. The BATC researchers built the Sensor Grid Testbed functionality on top of the Sensor Grid Middleware and built the application components required for the LVC SIDFOT program.

This Final Report documents the research completed on the AMSA TO 4 effort, along with lessons learned, recommendations and conclusions.

2.0 Introduction

2.1 Advanced Technology for Sensor Clouds Research Overview

As the 21st century warfighter becomes increasingly dependent on improved situational awareness while operating in complex urban environments, there will be an increased use of multi-layered sensor systems that include public and commercial sensors with sensor image processing capabilities provided through cloud computing services. These trends have given rise to the need to research not only the vulnerabilities of these public and commercial sensors, but also the need to investigate techniques and methods for alerting multi-layered sensor system users when cyber attacks have occurred. In addition to the fundamental sensor cloud research and corresponding results obtained from this effort, the Sensor Grid Testbed will provide the Distributed Collaborative Sensor Systems Technology Branch (AFRL/RYWC) and the Trusted Avionics Systems Network Branch (AFRL/RYWB) of the Integrated Electronic & Net-centric Warfare Division (AFRL/RYW) with a venue to further explore sensor grid vulnerabilities, as well as develop trust and trustworthiness algorithms for alerting Global Information Grid (GIG) users of cyber attacks.

2.2 Program and Project Objectives

This task order is part of the larger AMSA research program that encompasses a broader scope of R&D focused on trustworthiness research in accordance with the AMSA contract Statement of Work (SOW). The overall objective of the broader AMSA program is to research, develop, and demonstrate advanced (evolutionary and revolutionary) technologies integral to building an integrated information and knowledge-centric testbed for exploring the development of decision support solutions for 21st century warfighter operational challenges.

The more focused objective of this AMSA TO 4 effort was to conduct research, develop technology and components, and integrate the results for prototyping scalable cloud computing and advanced sensor management services into a Multi-Layered Sensor Grid testbed. The intent of the Sensor Grid testbed is to provide a framework for conducting advanced trustworthiness related research associated with network centric operations in urban operation scenarios. The specific purpose of this research was to leverage the design and development of Multi-Layered Sensor Grid technologies accomplished under related AFRL/RYW research efforts. These related efforts encompassed research that prototyped next generation technologies for integrating and facilitating sensor interoperability, data-mining, geographic information system (GIS) and archiving grids using publish-subscribe based mediation services. Using the results from the earlier research, the AMSA TO 4 research team was able to investigate the incorporation of cloud computing technologies and examine the penetration vulnerabilities of these technologies. Specifically, this research leveraged and used prototypes developed for the Sensor-Centric Collaboration Grid Middleware Management System (SCGMMS) with User-Defined Operational Picture capability (UDOP) and a Community Collaboration Grid Building Tool.

Coincidental with the start of this research effort, another cross-directorate research program entitled LVC SIDFOT was started that would leverage the results of the AMSA TO 4 research. While the initial intent was to have the Sensor Grid testbed completed prior to the start of the LVC SIDFOT program, a delay in the start if AMSA TO 4 prevented this from happening. Nevertheless, LVC SIDFOT requirements were used to influence the development of a realistic operational scenario application, including sensor interfaces and clients. While evolving during the course of this research effort, the Sensor Grid testbed prototype was also used on the LVC SIDFOT program. At the conclusion of the AMSA TO 4 research, the Sensor Grid testbed was delivered in place to continue supporting the LVC SIDFOT cross-directorate program.

2.3 Document Overview

This document covers the overall AMSA TO 4 accomplishments in the main task areas of AMSA development and AMSA support research. Under the AMSA TO 4 effort, specific research and development was performed in the areas of (1) Enhancing the Core Infrastructure Sensor Grid Middleware, (2) Developing an Enhanced Sensor Grid Application, (3) Researching and Implementing Trustworthiness Algorithms, and (4) Prototype Development, Integration, and Demonstration.

This document provides a detailed discussion of the research accomplishments achieved under this TO. The document also includes lessons learned from performing this research effort, recommendations for future efforts, and conclusions related to the research performed on this TO.

3.0 Methods, Assumptions, and Procedures

3.1 Background

Grid computing continues to evolve into cloud computing where real-time scalable resources are provided as a service over a network or the Internet to users who need not have knowledge of, expertise in, or control over the technology infrastructure ("in the cloud" as an abstraction of the complex infrastructure) that supports them. A sensor network can be a wired or wireless network consisting of spatially distributed autonomous devices using sensors to cooperatively provide data from different locations and multiple altitudes if appropriate. A sensor grid integrates multiple sensor networks with grid infrastructures to enable real-time sensor data collection and the sharing of computational and storage resources for sensor data processing and management. It is an enabling technology for building large-scale infrastructures, integrating heterogeneous sensor, data and computational resources deployed over a wide area, to undertake complicated tasks such as Intelligence Surveillance and Reconnaissance (ISR). A sensor web is an amorphous network of spatially distributed sensor platforms that synchronously communicate and are router-free. Cloud computing offers an expanded avenue for sensor webs to evolve into sensor clouds. There is a growing need for multi-layered sensing with hundreds, even thousands of sensors providing their inputs as military operations require increased, real-time situational awareness in a variety of operational scenarios. The real-time monitoring, processing, exploiting and analyzing of a large number of sensors and the corresponding data is quickly giving rise to the need for a better understanding of sensor clouds, multi-layered sensor system vulnerabilities and research into the trust and trustworthiness of these complex systems.

During previous research efforts, the development of the SCGMMS capability included the Net-Centric Collaboration Grid Middleware, the Collaboration Community Grid Builder and a UDOP tool, as well as a Common Operations Picture (COP) Tool. It used the Grid of Grids architecture as a baseline to prototype a set of Net-Centric Enterprise Services (NCES) as the Core Enterprise Services. SCGMMS is an Extensible Collaborative Sensor-Centric Grid Framework that supports UDOP/COP using a Sensor as a Service implementation mechanism.

This Advanced Technology for Sensor Clouds research supports the overall research focus in the Sensor Directorate's AFRL/RYW to conduct basic research, exploratory and advanced development programs to develop and deliver integrated electronic and net-centric warfare technologies and systems. Furthermore, the research helps to demonstrate autonomic, distributed, collaborative, and self organizing systems for integrated electronic and net-centric warfare. It also enables the government personnel to develop trusted avionics system architectures in support of secure and assured information operations. In particular, it helps with demonstrating sensor system data links necessary to connect both tightly and loosely coupled sensing oriented architectures’ to modern service oriented architectures in support of electronic and net-centric warfare. The results of the research also helps with performing system vulnerability analysis supporting integrated electronic and net-centric warfare technology development.

Furthermore, the result of this research supports the AFRL/RYWB mission related to conducting basic research, exploratory and advanced development programs to plan, develop, validate and deliver secure avionics and related system bus/network/backplane integration technology. It also supports the development and demonstration of system bus and associated data link technology to facilitate reliable trusted system interactions. It helps AFRL/RYWB personnel to conduct research and development in technologies supporting identification and tracking of threats to integrated weapon system networks. Finally, it helps with the overall performance of research and conducting avionics system experimentations and demonstrations for integrated system vulnerability analyses with an end to isolate and protect blue systems.

And finally, the result of this research supports the AFRL/RYWC mission related to conducting basic research, exploratory and advanced development programs to plan, develop, validate and deliver autonomic, distributed, collaborative sensor systems and sensor grids in support of electronic combat, support, and protection technologies. The resulting Sensor Grid Testbed enables the AFRL/RYWC personnel to perform basic research, exploratory and advanced development of predictive analytic technologies to optimize objective driven, self-organizing, collaborative sensor systems and generate anticipatory intelligence from vigilant sensing. Furthermore, it helps them to perform research and development in technologies supporting identification and tracking of critical sensor knowledge in self-organizing sensor webs.

The research associated with this task order effort investigated architectures for convenient scalable deployment of large sensor networks to support trusted sensor grids including sensor clouds, enhanced fault tolerance, and extended sensor management services. The outcome of the task is technical research documented in presentations, an enhanced prototype sensor grid testbed, and a final report addressing the subjects as identified below.

3.2 Research Overview

The research associated with this task order focused on investigating, developing, evaluating, and integrating the appropriate sensor grid technologies required to extend the SCGMMS for the new sensing technologies required for future sensor directorate research. The following subtasks outline the research activity that was accomplished to support this task order.

3.2.1 Enhance Core Infrastructure Sensor Grid Middleware

The AMSA TO 4 research team worked with AFRL personnel to determine the enhancements for the next generation Sensor Grid based on previous AFRL accomplishments and future research efforts. Extensions considered included new cloud computing technology to enable convenient scalable deployment for large sensor networks for Department of Defense (DoD) and commercial use; new services from the Open Grid consortium and other open system architectures, industry standards; and new technology capabilities needed for potential operational use.

3.2.2 Develop Enhanced Sensor Grid Application

The AMSA TO 4 research team developed and implemented some illustrative military applications of the enhanced Sensor Grid in a multi-layered sensing urban scenario involving defense related, homeland security, and commercially available sensors. Some considerations for inclusion in the military application were a capability to dynamically task and configure groups of sensors for selected layered sensing architectures. Additionally, applications that could link to other AFRL research on trust in complex systems were given a high priority. The former considerations included applications related to services for trusted data exchange among heterogeneous devices, secure connectivity across multiple sites, and the ability to depict trust within the COP. As part of coordinating with AFRL personnel, the determination was made to ensure that the key focus of an enhanced sensor grid application was to feed technology, and support the overriding requirements of the cross-directorate LVC SIDFOT program.

3.2.3 Research and Implementation of Trustworthiness Algorithms

The AMSA TO 4 research team also worked with AFRL personnel to research, assess, and evaluate possible collective trust algorithms and services that use cross validation to enhance trust and concatenate security, reliability, and other data from sensors. Specifically, this research activity extended previous research efforts associated with a database of trust metrics and analysis services for current and projected trust estimates.

3.2.4 Prototype Development, Integration, and Demonstration

Finally, the AMSA TO 4 research team integrated the methodologies, technologies, and software resulting from the above sub-tasks into an enhanced Sensor Grid Testbed prototype. After consulting with the AFRL/RYWC personnel, it was determined that instead of installing the Sensor Grid Testbed prototype into an AFRL/RYWC facility, the Sensor Grid Testbed would be delivered in place and incorporated into the LVC SIDFOT system for use in the cross-directorate research program. The capabilities of the Sensor Grid Testbed, including the sensor cloud, was demonstrated to show a multi-layered sensing urban scenario involving defense related, homeland security, and commercial sensors.

3.3 Research Terminology, Methodology and Approach

The emergence of cloud technology has raised a renewed emphasis on the issue of scalable on-demand computing. Cloud back-end support of small devices such as sensors and mobile phones is one important application. A preliminary study completed by the Anabas and IU members of the research team reports measured characteristics of distributed cloud computing infrastructure for collaboration sensor-centric applications on the FutureGrid [1, 2]. The study describes the team's understanding of the characteristics of the underlying network and its impact on multipoint, distributed cloud scalability. The report includes findings in areas of performance, scalability and reliability at the network level using standard network performance tools. The research team has also measured data at the message level using the NaradaBrokering (NB) system [3-8] by the Indiana University Community Grids Laboratory which supports a large number of practical communication protocols. Results are also presented at the collaboration and communication applications level using the Anabas sensor-centric grid framework [9], a message-based sensor service management and sensor-centric application development framework.

Geographically distributed and heterogeneous clouds in the FutureGrid are used because of their support for scalable simulations. The preliminary data indicates that a heterogeneous cloud infrastructure like FutureGrid coupled with a flexible collaborative sensor-centric grid framework is suitable for the study and development of new, scalable, collaborative sensor-centric system software and applications.

Some technical terms could have different meaning when used by researchers in different communities or applications. This is particularly evidential in inter-disciplinary and emerging fields. For clarity and consistency, several key terminologies used throughout this report are discussed next.

For this report, collaboration is defined as the general sharing of digital objects, and a sensor broadly as a source of a time-dependent stream of information. And the definition of real-time is application-specific. In the case of a voice over internet protocol (VoIP) application, for instance, a round-trip latency of less than 300 milliseconds is considered acceptable timeliness while other collaborative applications could have more stringent real-time requirements. Grids have been extensively discussed in the literature. For this report grids represent the system formed by the distributed collections of digital capabilities that are managed and coordinated to support some sort of enterprise [11]. Clouds are commercially supported data-center models competing with compute grids and general-purpose computing centers [12]. Clouds do not supplant data grids.

In an earlier study of collaborative applications [10] on Amazon’s Elastic Compute Cloud (EC2), research team members devised a methodology to study the characteristics of distributed cloud computing infrastructure at the network, transport messages, and message-based collaboration applications levels. They were able to measure performance at the network layer and modeled typical multipoint VoIP application-level traffic at the transport layer. The researchers had access to two clouds only, those at the Amazon EC2 US-East and Europe-West.

This research team adopted the same methodology in the study on FutureGrid. However, several significant differences exist between the study on the FutureGrid and that on the Amazon EC2. In the Future Grid study, researchers were able to conduct performance measurements on the network, transport messages, and message-based collaboration applications levels. They also extend their experiments on a homogeneous, 2-point, EC2 clouds to a heterogeneous, 4-point, Nimbus and Elastic Utility Computing Architecture Linking Your Programs To Useful Systems (Eucalyptus) clouds.

The overall approach for the accomplishment of the AMSA TO 4, Advanced Technology for Sensor Clouds, research was to investigate and implement enhancements to the core infrastructure Sensor Grid Middleware. This included the investigation, development and integration of several innovative technical capabilities into the core infrastructure sensor grid middleware, and delivery of several sensor grid middleware iterations for use in the development of the Sensor Grid Testbed to be used in the LVC SIDFOT program.

4.0 Results and Discussion

4.1 Enhance Core Infrastructure Sensor Grid Middleware (SensorCloud Architecture)

4.1.1 Sensor Cloud Overview

4.1.1.1 Introduction

The Sensor Cloud middleware is based on the Sensor Centric Grid Middleware Management System created by Anabas, Inc. The research team extends a sincere thanks to Anabas, Inc. and its CEO, Alex Ho for developing the SCGMMS and acknowledge that many of the ideas in this section are based on that work.

The objective of the Sensor Cloud Project was to provide a general-purpose messaging system for sensor data called the *Sensor Grid Server,* and a robust *Application Program Interface (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 was accomplished by implementing the *publish/subscribe* (pub/sub) design pattern which allows for loosely-coupled, reliable, scalable communication between distributed applications or systems.

4.1.1.2 Publish/Subscribe Architecture

The pub/sub design pattern 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 a 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.

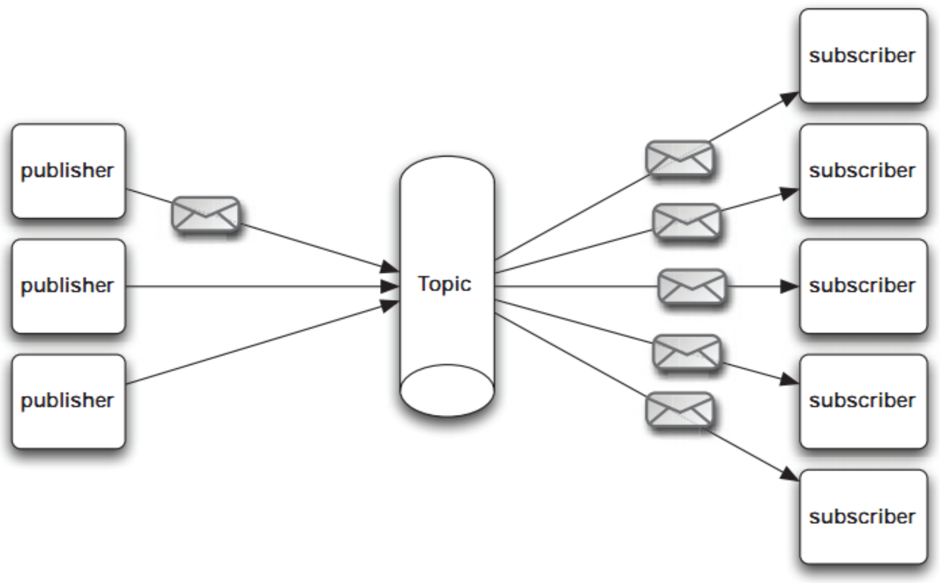


Figure 1. Elements of a Publisher/Subscribe System

A pub/sub system can be conveniently implemented using a Java Messaging Service (JMS) compliant Message-Oriented Middleware (MOM) such as NaradaBrokering, ActiveMQ, SonicMQ etc. to handle message mediation and delivery.

4.1.1.3 Sensor Cloud Overview

The Sensor Cloud implements the pub/sub 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

- Narada Broker is used as the default underlying MOM but any other JMS style broker could be used instead.

Figure 2 shows a high-level schematic 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*.

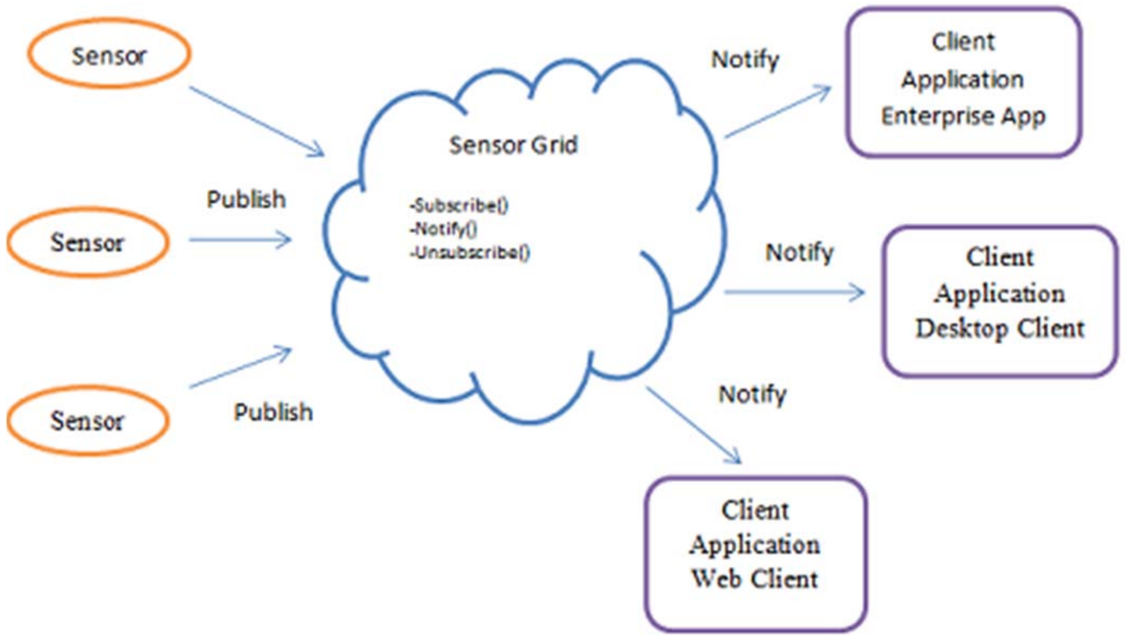


Figure 2. Schematic of the Sensor Cloud

Examples of physical devices implemented at the outset of this research include:

- Web/IP Cameras
- Wii Remotes
- Lego MindStorm NXT Robots
- Bluetooth Global Positioning System (GPS) Devices
- Radio Frequency Identification (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.

4.1.1.4 High-Level Sensor Cloud Architecture

One of the key goals of the Sensor Cloud Project was 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). NB is the transport-level messaging layer for the Sensor Grid. The overall Sensor Grid architecture concept is shown in **Figure 3**.

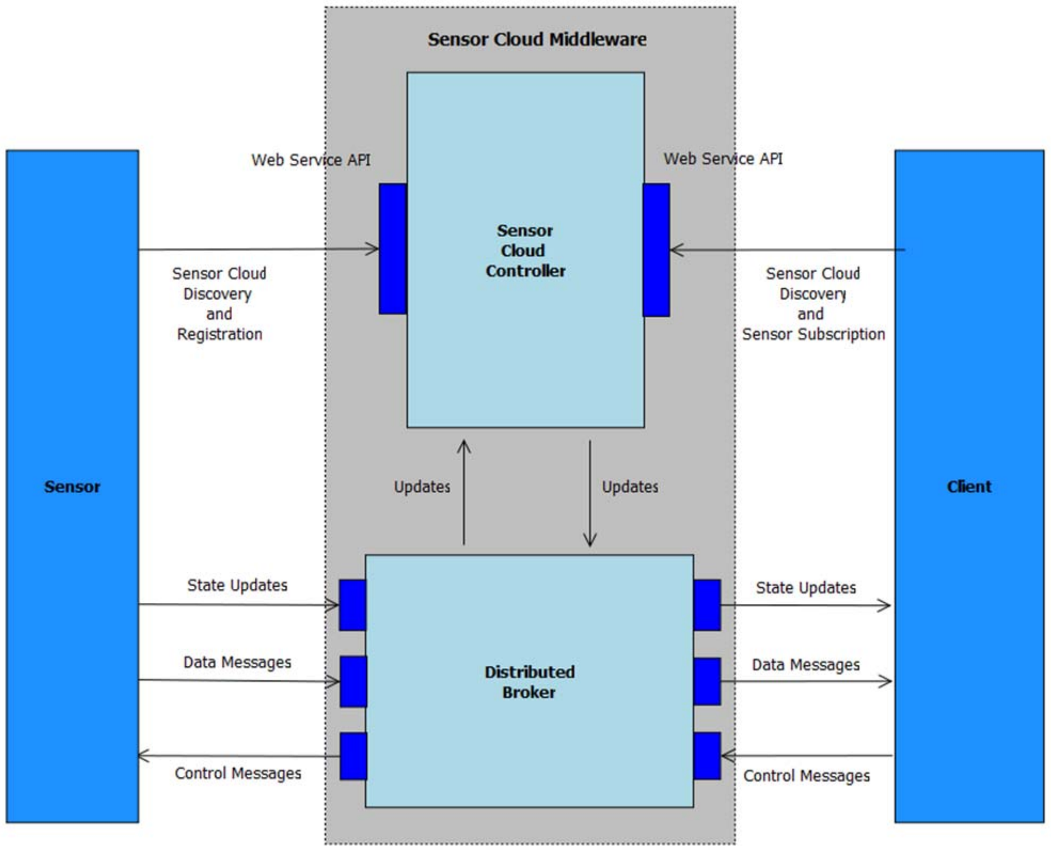


Figure 3. Sensor Grid Components

As this research matured and the Sensor Grid software was ported to a cloud, the terminology of Sensor Cloud and Sensor Grid are used somewhat interchangeably. Hence, the Sensor Cloud Middleware shown in Figure 3 is also called Sensor Grid Middleware. Furthermore, the Sensor Cloud Controller in Figure 3 is also interchangeable with Sensor Grid. And finally, the Web Service API shown on the right side of Figure 3, interfacing to the blue Client Box is also called the Application API; and the Web Service API shown on the left side of Figure 3 is also called the SSAL .

4.1.1.5 Sensor Grid Server (SG)

The SG mediates collaboration between sensors, clients (or applications) and the Grid Builder (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 through the application API, which in turn communicates with the 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 forwarding 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 the application if there are any changes for the relevant sensors.
- Sensor - Sensors' properties are defined by each sensor itself. Applications have to obtain this information through SG.
- Application/Sensor flow – The SG provides each application with the sensor information that it needs according to the filtering criteria. The application then communicates with sensors through the application API for receiving data and sending control messages.

4.1.1.6 Application API

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

4.1.1.7 Sensor

As noted earlier, 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., Really Simple Syndication (RSS), Web page) or task-oriented Computational Service (e.g., video processing service).

4.1.1.8 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 the SCMW. 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 SSAL.

4.1.2 Sensor Cloud Middleware

As noted earlier while discussing the SG in general, the SCMW Management System 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 Application API, while many of 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 SCMW's SSAL (see Section 4.1.5.2.2 for details).

NB is the underlying transport-level messaging layer for SCMW. It is a distributed message-based transport network based on the pub/sub messaging model.

By using NB as the transport layer, different components of SCMW can be deployed and work collaboratively in a distributed manner.

The overall architecture of SCMW is shown in **Figure 4**. Internally SCMW is composed of two main modules—SG and GB—which serve different functions. The major elements of the SCMW pictured in **Figure 4**, and the data flow associated with these elements, are discussed in section 4.1.2.1 through 4.1.2.5.

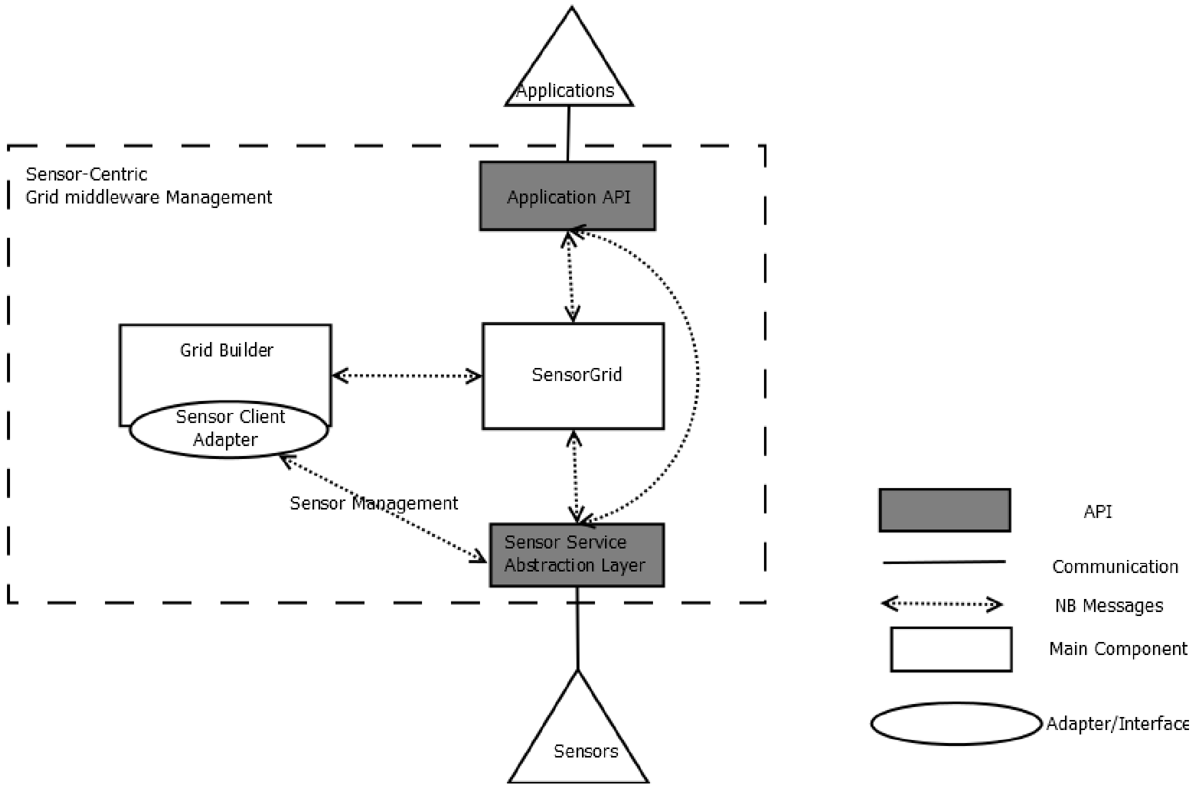


Figure 4. Sensor Cloud Middleware

4.1.2.1 Grid Builder (GB)

Given the large amount of sensors, GB is a sensor management module which provides mechanism and services to do the following:

1. Define the properties of sensors.
2. Deploy sensors according to defined properties.
3. Monitor deployment status of sensors.
4. Remote Management – Allow management irrespective of the location of the sensors.
5. Distributed Management – Allow management irrespective of the location of the manager / user.

GB itself posses the following characteristics:

1. Extensible – the use of Service Oriented Architecture (SOA) to provide extensibility and interoperability.

2. Scalable – management architecture should be able to scale as number of managed sensors increases.
3. Fault tolerant – failure of transports OR management components should not cause management architecture to fail.

The details of GB are discussed in Section 4.1.3.

4.1.2.2 Sensor Grid (SG)

SG communicates with a) sensors, b) applications, and c) GB to mediate the collaboration of the three parties. Primary functions of SG are to manage and broker sensor message flows.

4.1.2.2.1 Sensor/Sensor Grid flow

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 the SG except when it intentionally has to be recoded. In this case, SG will subscribe to data of that particular sensor.

4.1.2.2.2 Application/Sensor Grid flow

Applications communicate with SCMW through the 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 forwarding 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 applications if there are any changes for the relevant sensors.

4.1.2.2.3 Grid Builder/Sensor Grid flow

Sensors' properties are defined in GB, and applications obtain this information through SG. Moreover, filtering requests are periodically sent to GB for updating the lists of sensors needed for each application according to their defined filter parameters. Much of the information will be stored in a SG to minimize queries to GB.

4.1.2.2.4 Application/Sensor flow

SG provides each application with sensor information that the application needs according to the filtering criteria. The application then communicates with sensors through the Application API for receiving data and sending control messages. The details of SG are discussed in Section 4.1.4.

4.1.2.3 SCMW API

As noted earlier, the SCMW supports user and service provider applications for multiple domains (e.g., financial, military, logistics, aerospace etc.). SCMW provides an application API which allows any kind of application to retrieve information from the sensor pool managed by SCMW. The API also provides a filtering mechanism which provides applications with sensors matching only their designated querying criteria. Details of the SCMW API are discussed in Section 4.1.5.1.

4.1.2.4 Sensor

The sensor component of the SCMW is described in section 4.1.1.7.

4.1.2.4.1 Sensor Client Program

As noted earlier for the SG Server, an SCP is also required to connect to SCMW. In the case of the SCMW, SCP communicates with SCMW through SSAL (refer to Section 4.1.5.2 for details). **Figure 5** shows a physical sensor and the corresponding structure of the component compromising the SCP.

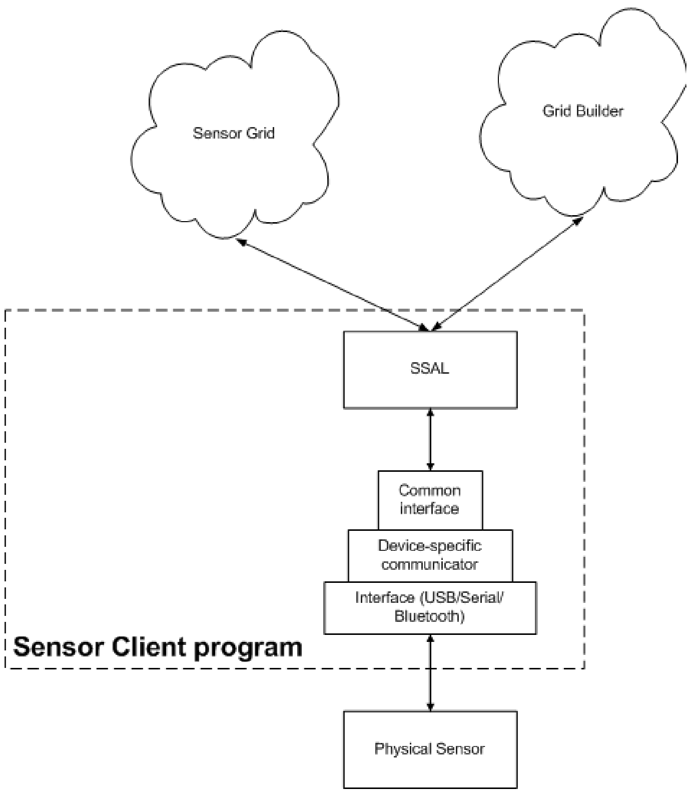


Figure 5. Structure of a Sensor Client Program

4.1.2.4.2 Computational Service

Computational Service is a special kind of sensor which does not take input from the environment. Instead, it takes the output of other sensors as its input, performs various computations on the sensor data, and outputs the processed data. Since a Computational Service also produces a time-dependent stream of data it matches the definition of a sensor.

Figure 6 shows the data flow of how environmental data is transformed by processing data through a sensor and a Computational Service. The architecture of SCMW allows the data source to be assigned and reassigned dynamically.

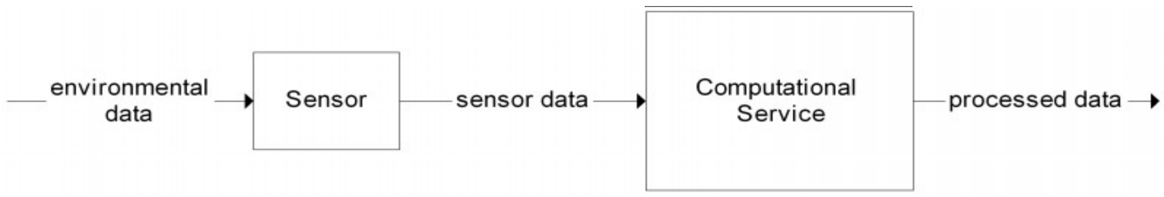


Figure 6. Computational Service

4.1.2.5 Sensor Service Abstraction Layer (SSAL)

SCMW can potentially support large number of different sensor types. Ease of adding new sensors by different sensor developers without internal knowledge of SCMW is one of the most important requirements. SSAL provides a common interface for adding new sensors to the system easily. Sensor developers can write simple programs utilizing SSAL libraries for connecting sensors to SCMW. Afterwards, the sensor will be available for all applications immediately. A more technical description of the SSAL is discussed in Section 4.1.5.2.

4.1.3 Grid Builder (GB) Design Discussion

4.1.3.1 Grid Builder Architecture Overview

Figure 7 depicts the top-level overview of the GB architecture. GB is originally designed for managing Grid-of-Grids. For this project, GB was extended to include the management of a generalized sensor-centric grid of grids. The following discussion of GB will focus on this specialized version. CGL-developed hpsearch is adopted and extended for this work [20].

The Grid which GB manages is arranged hierarchically into Domains. Each domain is started by its Bootstrapping Service and is typically, but not necessarily, a single PC which manages sensors which are closely related. Sensors can be deployed from any PC which is accessible from one of the domains. There can be only one root node in the grid known as the Root Domain. Within each domain, there exist some basic components as described next.

Managers and Resources

GB manages grids and resources through a manager-resource model. Each type of Resource which does not have a Web Service interface should be wrapped by a Service Adapter (SA). Each kind of SA is managed by a corresponding Manager.

Since the grid contains sensors, a Sensor Manager is responsible for managing sensors through Sensor Service Adapters (SSA). Each SSA has its own set of defined Sensor Policy. This policy tells the Sensor Manager how the SSA is to be managed, and defines the properties of the sensor bound to the SSA.

Health Check Manager

The Health-check Manager is responsible for checking the health of the whole system (ensures that the registry and messaging nodes are up and running and that there are enough managers for resources).

Bootstrapping Service

This service ensures that bootstrap processes of the current domain are always up and running. For example, it periodically spawns a health-check manager that checks the health of the system.

Registry

All data about registered services and service adapters are stored in memory called Registry. Registry is used to process messages so it can manage new SA, renew SA and update SA status.

4.1.3.2 Significant Classes

4.1.3.2.1 Class Diagram

A detailed class diagram for the GB is shown in **Figure 8**. This class diagram shows all of the significant classes in the GB. They are categorized into five main categories: Messaging Layer, Domain Management, Managers, Resource Management, and Registry. Each of these class categories will be discussed in detail.

Messaging Layer

GB is built on top of a message-based architecture. All modules in GB such as BootstrapService, ForkDaemon, Managers, Registry and ServiceAdapters are standalone and communicate with one another by message passing. With this model, separate modules can be deployed as distributed services.

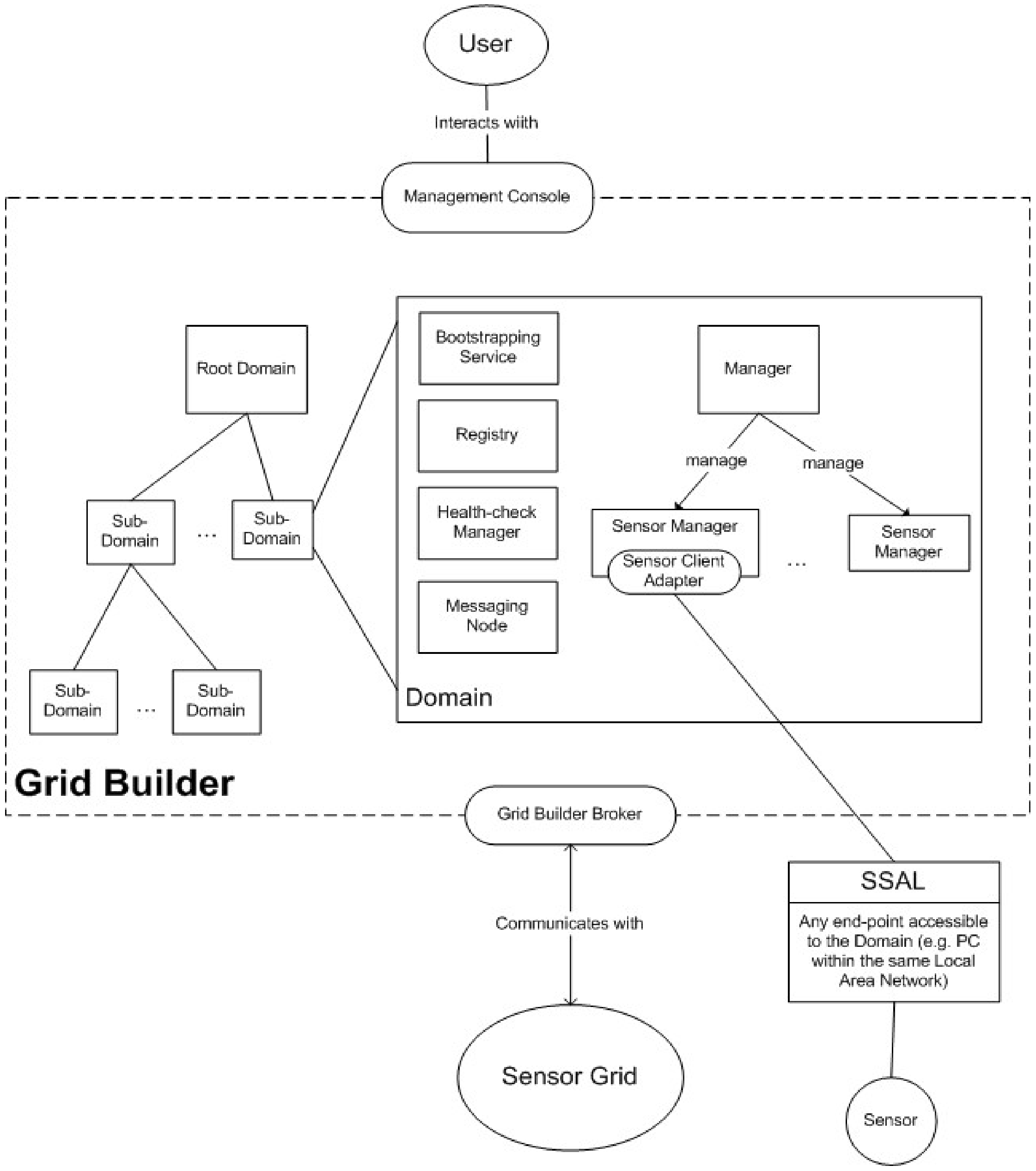


Figure 7. An overview of the Grid Builder architecture

GB has a set of classes dedicated for message passing. Each module has a unique universally unique identifier (UUID) and one or more Universal Locator(s) (UL). UL provides all the information necessary to identify a module in the network, including transport type, host address, port and path. Four transport types are supported: user datagram protocol (UDP), transmission control protocol (TCP), hypertext transfer protocol (HTTP) and NB. Each UL is responsible for message of one transport type.

TransportSubstrate is responsible for sending and receiving messages to and from a module. It automatically serializes the message content according to the transport type of destination. Once created, it spawns a thread which keeps waiting for incoming messages and notifies the associated MessageProcessor upon message arrival.

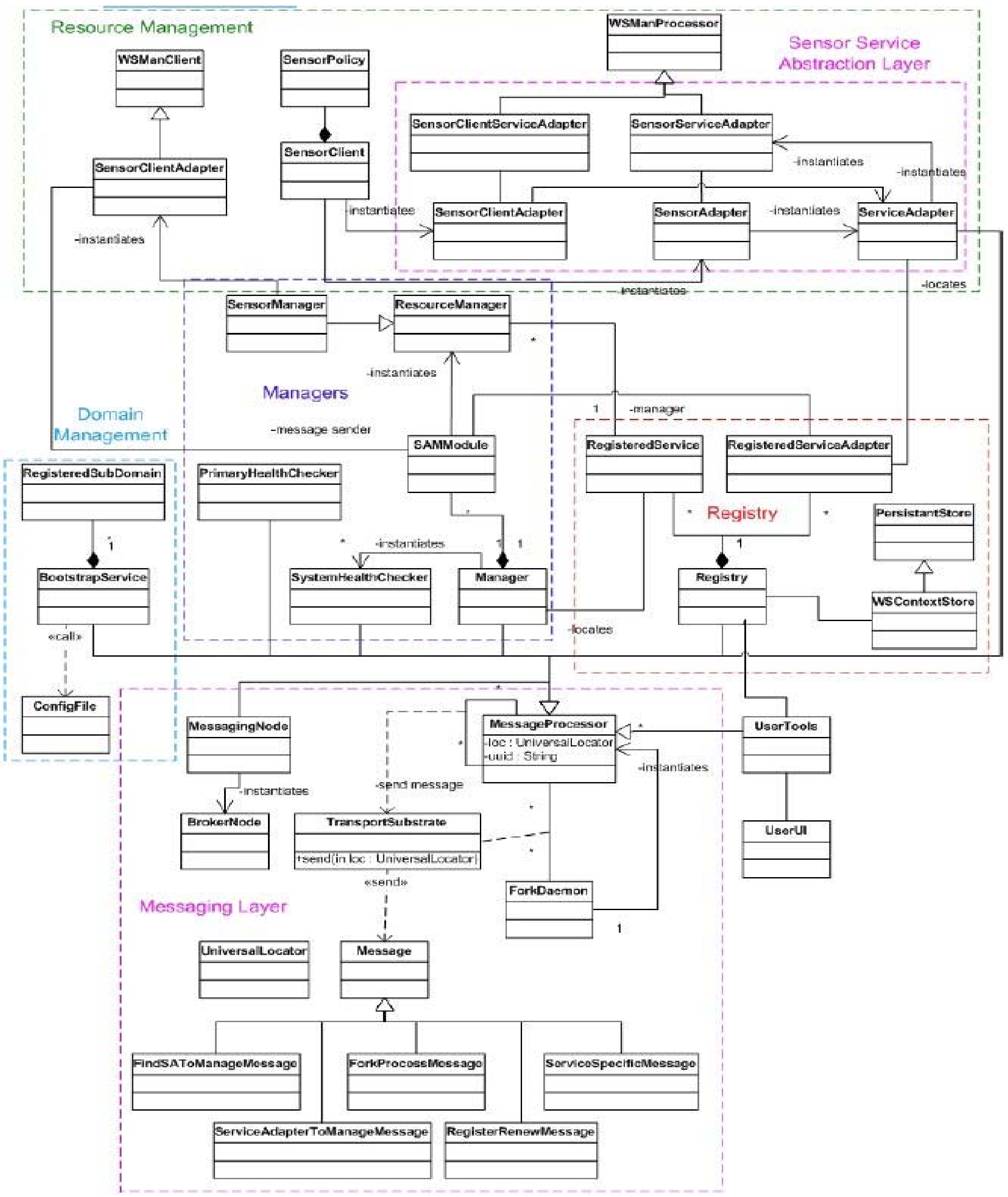


Figure 8. Class Diagram of Grid Builder

Modules which want to receive messages should implement the MessageProcessor interface and associate itself with a TransportSubstrate. Important modules which implement this interface include BootstrapService, Registry, SystemHealthChecker, Manager, ServiceAdapter and UserTools.

Communications between SensorManager and SensorServiceAdapters use the Web Service (WS) interface. WS in GB is built on top of this messaging layer.

Domain Management

Domain management in GB is done by BootstrapService. Each domain has one BootstrapService which constantly communicates with the BootstrapServices of other domains. Each domain hierarchy contains one Root node. Each domain connects with at most one parent node and any number of child nodes. For now the hierarchy is defined using a configuration file (mgmtSystem.conf). The operation flow of domain management is depicted in **Figure 9**.

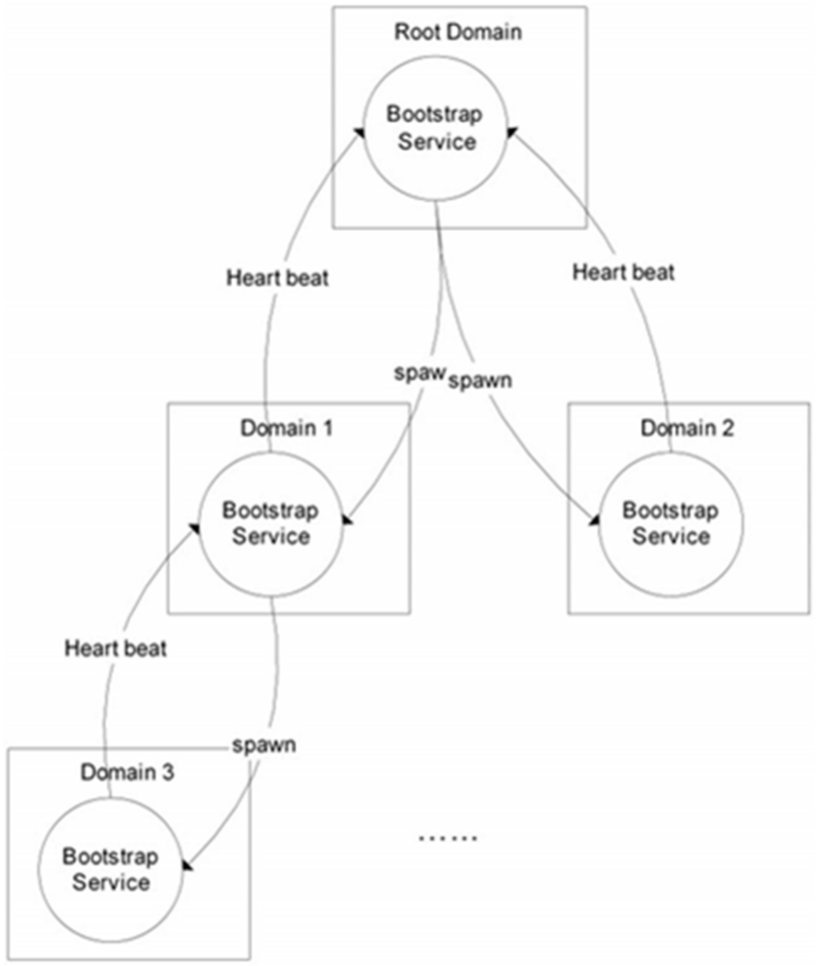


Figure 9. Domain Management

To keep the whole hierarchy up and running, each domain periodically sends a heart beat message to its parent domain. It also has to spawn the BootstrapService of all child domains if any of them is not sending a heart beat for some time.

Managers

In GB there are two levels of managers. The lowest level is ResourceManager, which manages resource specific modules. For example, SensorManager is responsible for managing a SensorServiceAdapter through the Web Service interface and performs operation such as sending policies to the adapters.

The upper level is Manager, which manages ResourceManagers and ServiceAdapters. The Registry keeps checking whether there are ServiceAdapters which have been registered but do not have a Manager during the health check sequence. If there is one, the Manager is notified and creates a SAMModule which in turn creates a ResourceManager for the particular resource in the ServiceAdapter. SensorClientAdapter is an adapter inside SensorManager for communication with the associated SensorServiceAdapter inside the Service Adapter. The interaction between the managers and service adapters is shown in **Figure 10**.

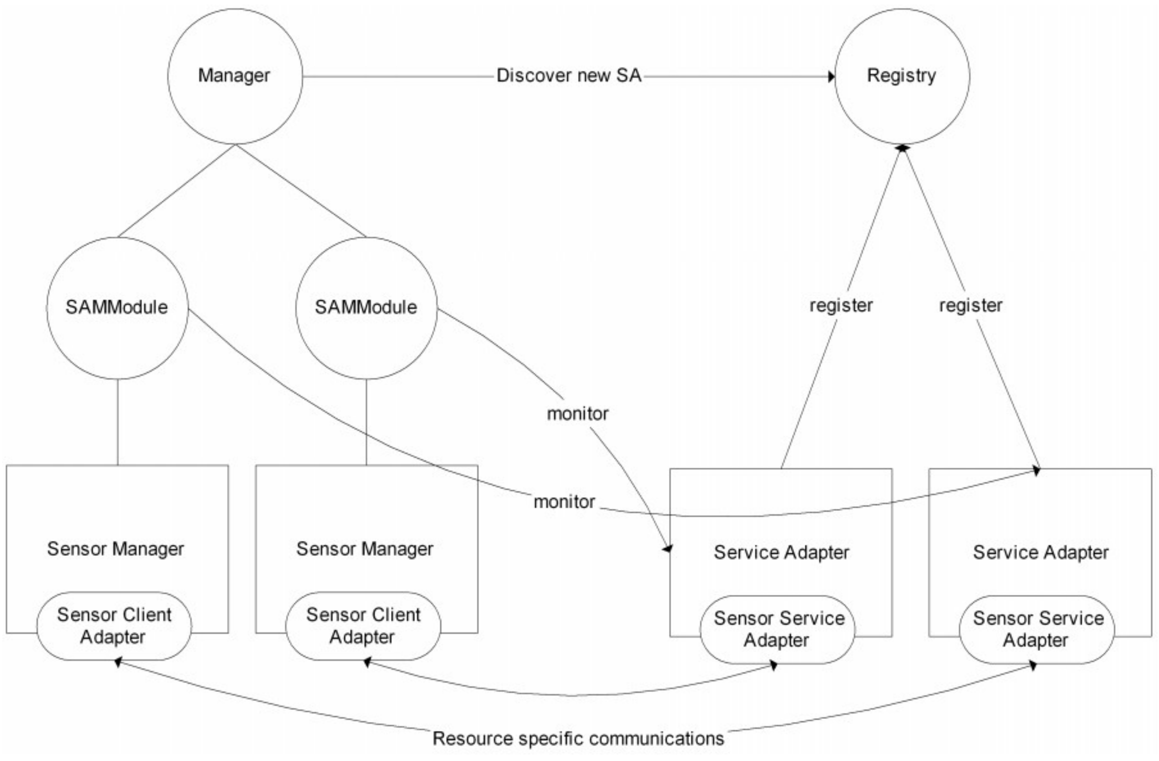


Figure 10. Manager and Service Adapter

Resource Management

These classes are at the resource level, where resource specific tasks are performed. Each sensor is treated as a resource in GB, and each sensor has a corresponding client program (represented by SensorClient) responsible for interfacing the sensor with SCMW.

SSAL is the interface for connecting all types of sensor client programs with GB. The class diagram only shows part of SSAL which resides in GB.

Communication between resource managers (i.e., SensorManager) and Resources (i.e., SSA) uses the WS interface for message passing. SSA therefore conforms to the WS "Put," "Get," "Delete," and "Create". "Get" is used for getting SensorPolicy of the sensor and initiates connection with SG. "Delete" is used for disconnecting connection with SG.

Registry

Each domain has a Registry which maintains the state of the entire domain, such as the Universal Locator of every module, how many Service Adapters have been registered, the status and policy of each sensor, which SA is assigned to which Manager, etc.

RegisteredServiceAdapter is a class which contains information of ServiceAdapter such as UniversalLocator, SensorPolicy and current status. RegisteredService contains information of non-SA modules such as Managers and MessagingNodes.

Registry can work with or without persistent storage. By default all information is stored in memory using hash tables. The user has an option whether to write all information to persistent storage so that it can be retrieved later on even if the domain is restarted. The persistent storage used is compliant to WS-Context specification [21].

Figure 11 shows the overall architecture of the Domains, Registry and WS-Context modules in Grid Builder. To use WS-Context, an Axis server and a MySQL server should be running in each domain for WS communication and storage. All domain related information in the Registry is stored in WS-Context and shared with other domains through NaradaBrokering's topic-based publish-subscribe messaging service.

Although the current implementation does not use WS-Context as a centralized database for service discovery, it can be easily enhanced to provide such service since the system is already WS compliant.

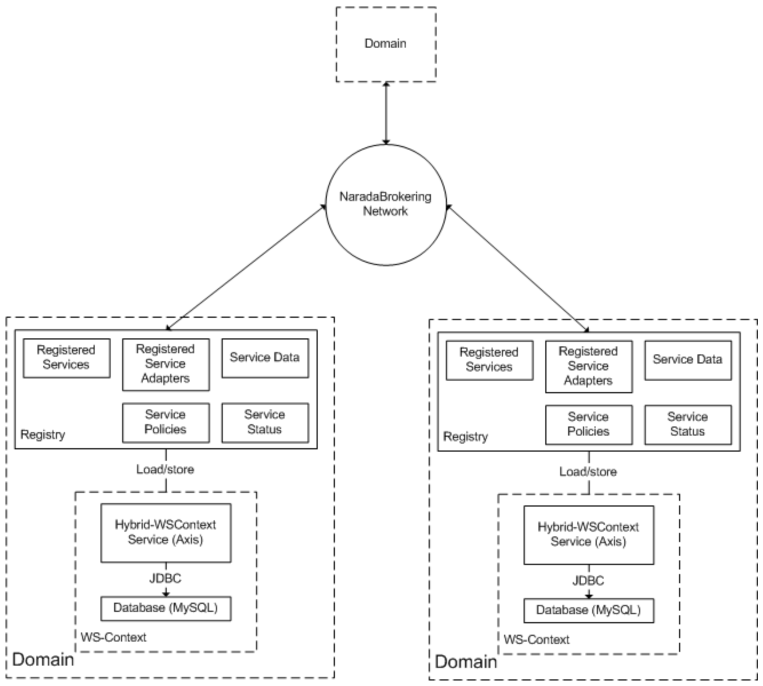


Figure 11. Registry and WS-Context

4.1.3.2.2 Class Description

This section provides brief description of each important class in GB.

Class name:	MessageProcessor
Package name:	cgl.hpsearch.core.transport
Description:	Interface for classes which use GB's messaging layer to receive messages
Important interface:	processMessage()

Class name:	MessagingNode
Package name:	cgl.hpsearch.core.services.messagingNode
Description:	Manages the GB's transport layer components (such as NB)
Important interface:	setBootstrapLocator(), startBrokerNode()

Class name:	TransportSubstrate
Package name:	cgl.hpsearch.core.transport
Description:	Responsible for receiving and sending messages to and from MessageProcessor using different transport protocols
Important interface:	register(), send(), getUniversalLocatorForTransport(), close()

Class name:	Message
Package name:	cgl.hpsearch.core.messages
Description:	Superclass of all types of messages in GB. Different types of message has different characteristics and serves different functions
Important interface:	getType(), getMessageId(), getTo(), getFrom(), getTimeStamp()

Class name:	UniversalLocator
Package name:	cgl.hpsearch.core.transport
Description:	A locator which lets different modules to identify one another for messaging passing. Records the host, port, and transport type of a module
Important interface:	getHost(), getPort(), getPath(), getTransportType()

Class name:	UserTools
Package name:	cgl.hpsearch.core.services.user
Description:	Responsible for forwarding different user operations (e.g., deploy sensors) to different modules in GB
Important interface:	getServiceData(), putServiceData(), retrieveStatus(), sendPolicyMessage(), sendRunMessage(), sendFilterMessage(), sendForkMessage()

Class name:	Manager
Package name:	cgl.hpsearch.core.services.manager
Description:	Manages all Resource Managers
Important interface:	processMessage(), startSAMManagementThread(), removeSAMManagementObject(), send()

Class name:	SystemHealthChecker
Package name:	cgl.hpsearch.core.services.manager
Description:	Responsible for checking whether all modules are up and running in a domain
Important interface:	processMessage()

Class name:	BootstrapService
Package name:	cgl.hpsearch.core.services.bootstrap
Description:	Responible for starting up all modules during domain initialization. Periodically spawns SystemHealthChecker and sending heart beat to parent domain

Class name:	ForkDaemon
Package name:	cgl.hpsearch.core.services.fork
Description:	Responsible for creating different modules locally as processes
Important interface:	process()

Class name:	SAMModule
Package name:	cgl.hpsearch.core.services.manager
Description:	Manages resources (sensors). Has one to one mapping to each Service Adapter and the corresponding Resource Manager.
Important interface:	send(), checkIfOwner(), getServiceData(), putServiceData(), spawnProcess(), sendMessage()

Class name:	SensorManager
Package name:	cgl.hpsearch.sensor
Description:	Resource manager for managing SensorServiceAdapter
Important interface:	processMessage(), getServicePolicy(), putServicePolicy(), runService()

Class name:	SensorClientAdapter
Package name:	cgl.hpsearch.sensor
Description:	The adapter of SensorManager for communication with SensorServiceAdapters using Web Service
Important Interface:	getServicePolicy, putServicePolicy(), runService()

Class name:	ServiceAdapter
Package name:	cgl.hpsearch.core.services.sa
Description:	Associated with a Resource Manager to manage the corresponding resource
Important interface:	start(), close(), publishData()
Class name:	SensorServiceAdapter
Package name:	cgl.hpsearch.sensor
Description:	Responsible for brokering the communication between a Resource Manager and sensor client program using Web Service
Important interface:	start(), close(), publishData(), handleSensorGridConnectionLoss(), setSensorProp(), processWxMGMT_Rename(), processWxfDelete(), processWxfPut(), processWxfCreate(), processWxfGet()
Class name:	SensorClientServiceAdapter
Package name:	cgl.hpsearch.sensor
Description:	Responsible for brokering the communication between a Resource Manager and service sensor client program using Web Service
Important interface:	start(), close(), publishData(), handleSensorGridConnectionLoss(), setSensorProp(), sendControl(), setFilter(), subscribeSensorData(), unsubscribeSensorData(), processWxMGMT_Rename(), processWxfDelete(), processWxfPut(), processWxfCreate(), processWxfGet()
Class name:	SensorPolicy
Package name:	cgl.hpsearch.core.policies
Description:	Holds resouce specific policy, that is the property of a sensor
Important interface:	getType(), getSensorProperty()
Class name:	WSManClient
Package name:	cgl.hpsearch.wsmgmt
Description:	Client interface for communicating with WSManProcessors (end points) using Web Service messaging
Important interface:	getMyEndPoint(), getServiceEndPoint(), setServiceEndPoint(), setWsEventingClient(), processMessage(), executeOneWay(), executeRequestReply(), sendOut(), CreateAndMarshallMessage()
Class name:	WSManProcessor
Package name:	cgl.hpsearch.wsmgmt
Description:	End point for receiving Web Service Message
Important interface:	setMessageSender(), setMyEndPoint(), processSOAPMessage(), processWxMGMT_Rename(), processWxfDelete(), processWxfPut(), processWxfCreate(), processWxfGet()

4.1.3.3 Important Features

4.1.3.3.1 System Health Check

Every module in GB is deployed in a distributed manager and linked together by different network protocols. A health check system is therefore fundamental to ensure every modules is indeed deployed and working properly. GB performs periodic System Health Check (SHC) to ensure that everything is up and running. SHC can be divided into three stages: Initialization, Detect Changes, and Maintain System State. These stages of operation will be described next.

Initialization

A block diagram of the SHC initialization is shown in **Figure 12**. To start a new Domain X, a user executes a script to perform a Primary Health Check Sequence. This action creates a Permanent Messaging Node, which is responsible for communication between all modules within a domain, and communication with other domains. After that, a Fork Daemon is created. Every module of Grid Builder (e.g., Registry, Service Adapters, Sensor Service Adapters, etc.) is executed as a separate process in the operating platform. Fork Daemon is responsible for creating modules as separate processes.

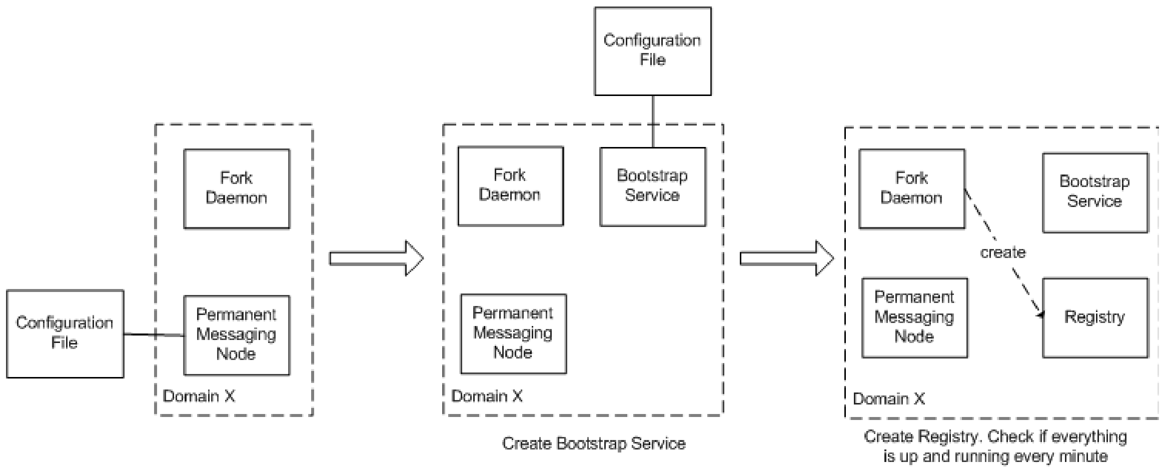


Figure 12. System Health Check (SHC) Initialization

After primary health check, the domain is now capable of receiving messages from other domains. The Bootstrap Service is launched when a message is received from the root domain. The Bootstrap Service is responsible for making sure that every module is up and running in a domain. It periodically spawns a System Health Checker to check the health of the system.

After Bootstrap Service has been initialized, it creates the Registry. The system then checks if all modules are up and running for every minute. If not, create the module that is missing (for details please refer to Section 4.1.3.4.3).

Detect Changes

When a user introduces a change (see **Figure 13**) to the system, such as deploying a sensor, SHC automatically detects and reacts to the change. For example, a user deploys a sensor by starting the corresponding sensor client program. The program automatically creates a new SA for the sensor which in turn creates an SSA. If no Manager is present in the domain, a Manager process is created by ForkDaemon to manage the sensor through the SA.

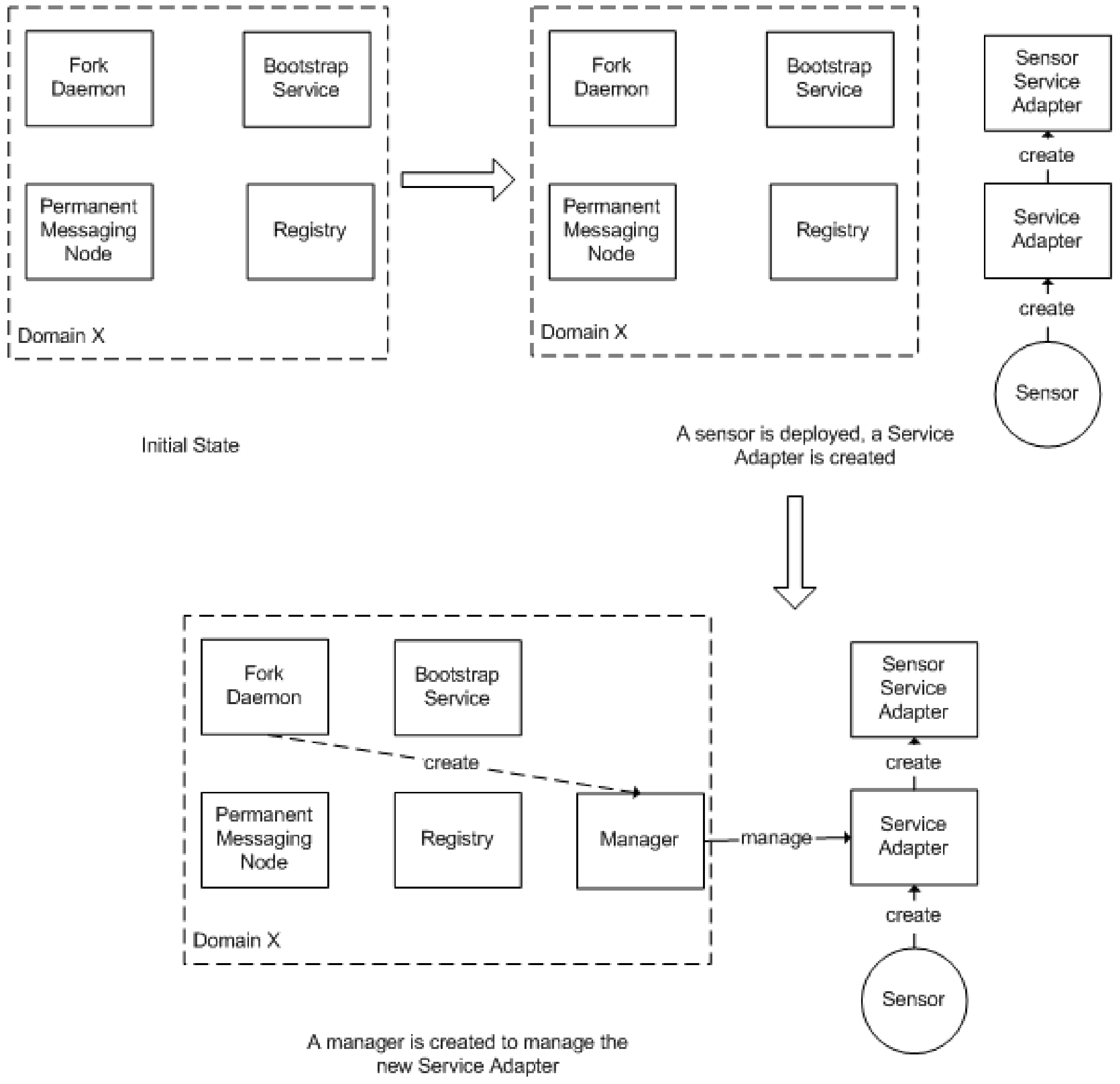


Figure 13. Adding Service Adapter

Maintain System State

To make sure that every resource is up and running, each module periodically notifies its manager and the registry of its presence (see **Figure 14**). This process implements the Maintain System state phase of operation.

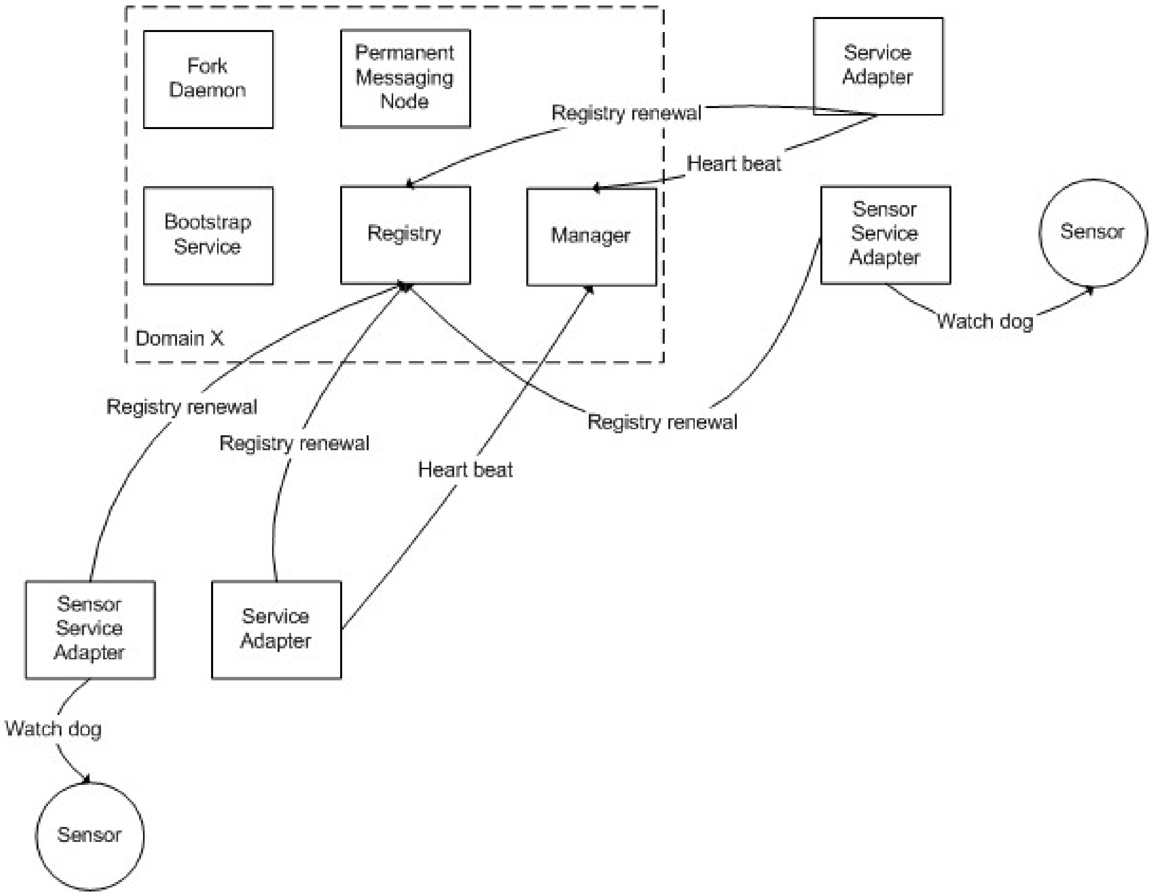


Figure 14. System Health Check (SHC) Maintaining System State

4.1.3.3.2 Classification Scheme

Classification defines all properties which are shared by all sensors supported by SCMW. Classification serves the following functions:

1. Allows GB to differentiate among different sensors for visualizing sensor's policies
2. Defines what can be filtered
3. Allows meaningful visualization of sensor data at application side
4. Allows application to differentiate different sensors

Figure 8 shows the class diagram of classification. It can be divided into 3 categories: Sensor Property, Sensor Data, and Message Sterilization. These categories will be described next.

Sensor Property

In order to introduce a new sensor to SCMW, several properties have to be defined in class SensorProperty. **Table 1** shows the sensor properties that need to be defined.

Table 1. Fields of Sensor Property

Property	**Description**
sensorId	A human readable ID for identification which does not have to be unique
groupId	Sensors can be assigned to different logical groups for easier management. GroupId identifies the group
sensorType	Textual description of the type of a sensor
sensorTypeId	An integer which helps identifying the sensor type. Application has to compare this together with field sensorType to uniquely identify the type of a sensor
location	Textual description of the location of a sensor, including street, city, state/province and country
historical	Defines whether to archive collected sensor data in SG. Currently this feature is not implemented
sensorControl	An array of integers which uniquely identifies each control message
controlDescription	A string array of textual description of control messages. Should align with sensorControl array
userDefinedProperty	A class which defines any user-defined properties specific for each type of sensor

SCMW comes with a set of predefined types. Class PredefineType contains information for generating predefined SensorProperty. UserDefinedProperty contains properties which are essential for the sensor but may not be common for all sensors (e.g., for deploying a RFID reader, it needs the COM port for hardware interfacing). A set of user-defined properties for predefined sensors are implemented as subclasses of UserDefinedProperty.

For location, class PredefinedLocation contains a list of predefined mapping of city names and GPS latitude-longitude for easy visualization on a map.

Sensor Data

For each type of sensor, its data format is usually quite different from other sensors. In SCMW (Class Diagram shown in

Figure 15), a class which extends SensorData should be created which defines how to decode and use data from a sensor.

Message Serialization

Each time before the property of a sensor is sent among modules (e.g., passing from GPSManager to SensorServiceAdapter and Registry), it is serialized into Extensible Markup Language (XML) format. Class SensorClassificationUtil provides operation for message serialization and deserialization.

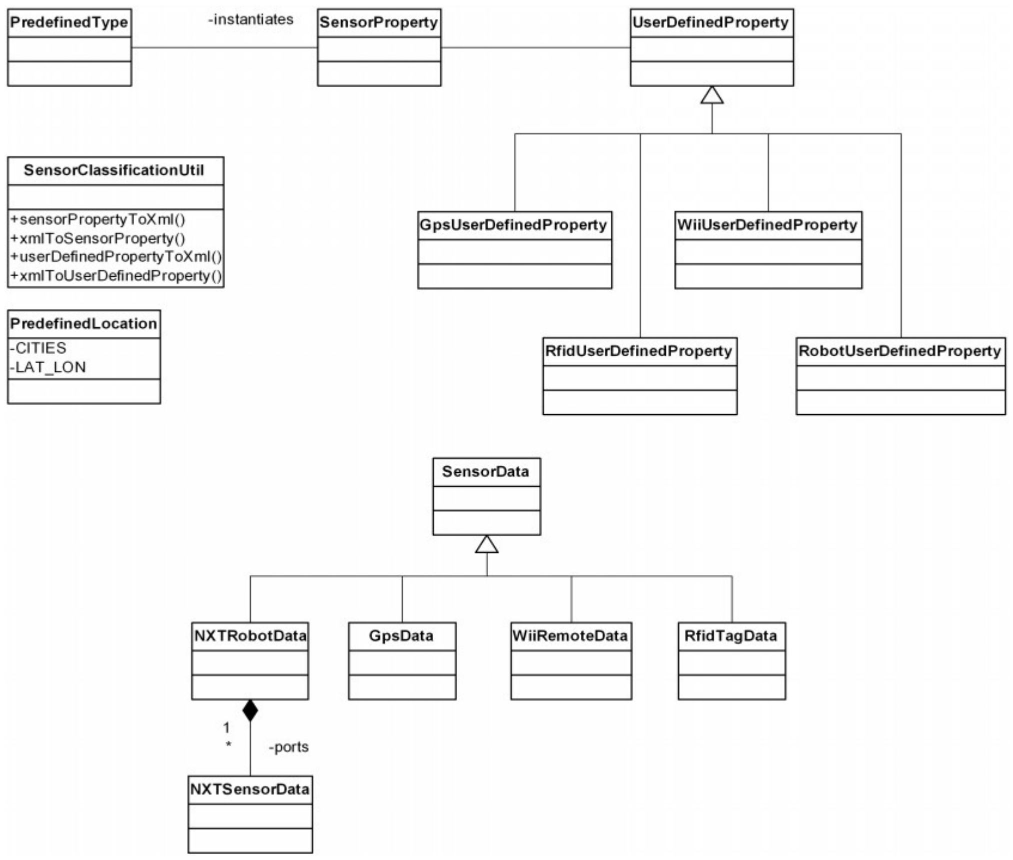
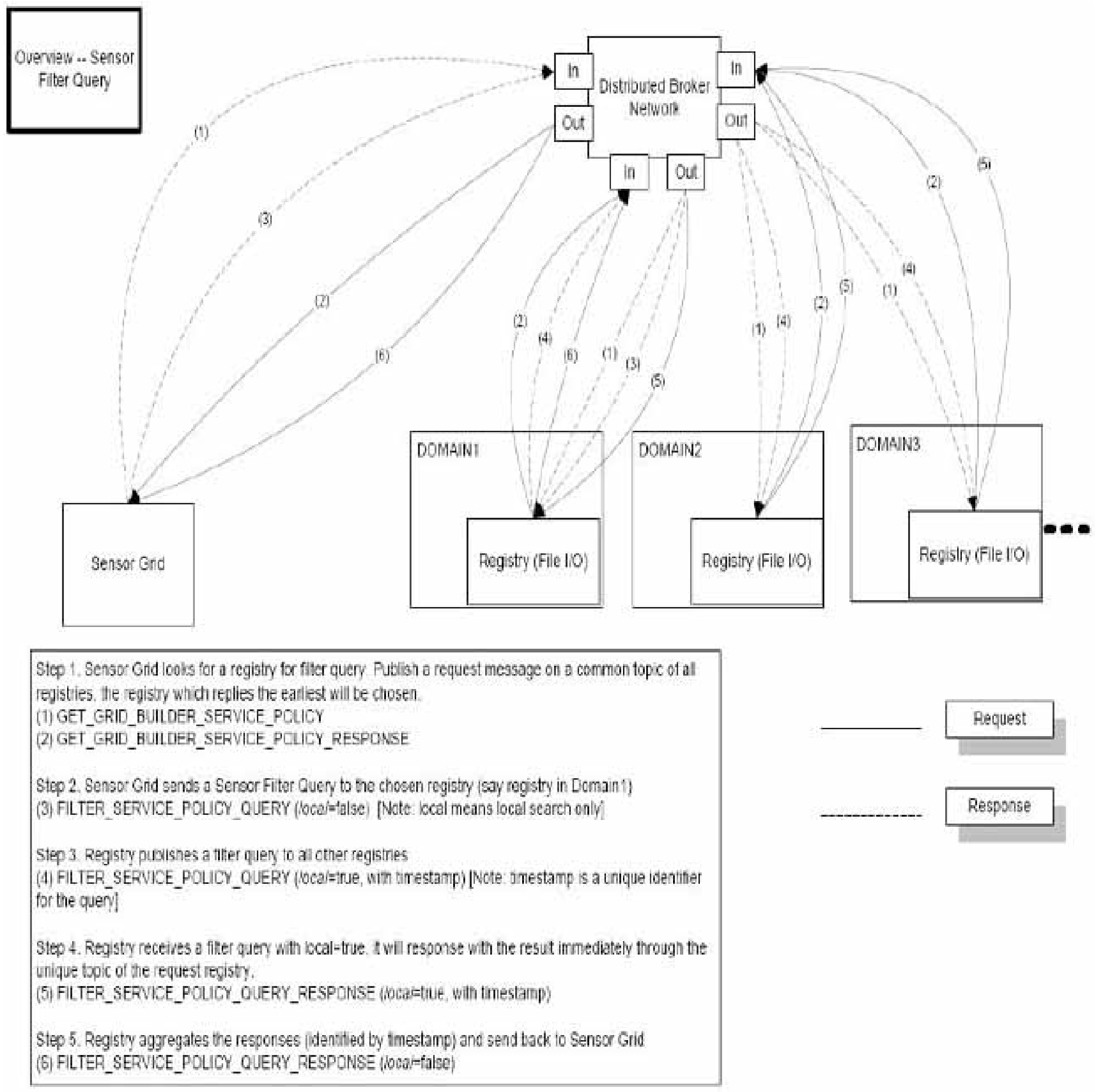


Figure 15. Class diagram of classification scheme in SCMW

4.1.3.3.3 Filtering Mechanism

At the application level, filtering is essential for retrieving only the required sensors from a possibly huge sensor pool. Filtering is done based on the SensorProperty of each sensor, which is defined according to “based on” rules in classification.

Figure 16 illustrates this concept.



Step 1. Sensor Grid looks for a registry for filter query. Publish a request message on a common topic of all registries, the registry which replies the earliest will be chosen.
(1) GET_GRID_BUILDER_SERVICE_POLICY
(2) GET_GRID_BUILDER_SERVICE_POLICY_RESPONSE

Step 2. Sensor Grid sends a Sensor Filter Query to the chosen registry (say registry in Domain1)
(3) FILTER_SERVICE_POLICY_QUERY (*local*=false) [Note: local means local search only]

Step 3. Registry publishes a filter query to all other registries
(4) FILTER_SERVICE_POLICY_QUERY (*local*=true, with timestamp) [Note: timestamp is a unique identifier for the query]

Step 4. Registry receives a filter query with local=true. It will response with the result immediately through the unique topic of the request registry.
(5) FILTER_SERVICE_POLICY_QUERY_RESPONSE (*local*=true, with timestamp)

Step 5. Registry aggregates the responses (identified by timestamp) and send back to Sensor Grid
(6) FILTER_SERVICE_POLICY_QUERY_RESPONSE (*local*=false)

Figure 16. SCGMlv.IS sensor filtering mechanism in a distributed architecture.

Defining a Filter

Applications have to define filtering criteria according to their UDOP requirements. The criteria are encapsulated in a SensorFilter object. A SensorFilter is composed of a set of properties defined in SensorProperty connected with Boolean "and" or "or" operators. Please refer to Section 4.1.3.3.2 for the definition of SensorProperty. Given that a list of sensor properties in a sensor filter are connected together with the "and" operator, only sensors which have properties that are an exact match in string comparison with ALL the properties defined in the filter should get through. Similarly sensors which have properties with an exact match in string comparison with ANY of the properties defined in a sensor filter with sensor properties connected together with the "or" operator should get through.

The list of "and" and "or" sensor properties are represented as a 2D string array in SensorFilter. For example, if someone wants to get a list of SA IDs which have policy ((sensorType=GPS and location="Hong Kong") or (sensorType=RFID and location="New York" and historical=true)), set the filter like this:

SensorFilter filter=new SensorFilter(); String[][] comp=new String[2][]; comp[0]=new String[2];

comp[1]=new String[3]; comp[0][0]="sensorType=GPS"; comp[0][1]="location=Hong Kong"; comp[1][0]="sensorType=RFID"; comp[1][1]="location=New York"; comp[1][2]="historical=true"; filter.setOrComparison(comp);

Data Flow

Filtering is done in three stages: Application to SG, SG to GB, and SG to application. These stages will be described next.

Application to SG

A filter query request is initiated from the application. For each filter query, fields which exist in SensorProperty can be combined using the "and" or "or" operator to form a query string. This string is then sent to SG.

SG to GB

SG forwards the request to GB. At this stage, GB searches through the registry of all domains and aggregates the unique id of sensors which match the query in a response message. The response message is then sent back to SG. SG periodically checks if the filter request from application changes. If it does, the application is notified in the same manner.

SG to application

SG releases the resources (e.g., unsubscribe sensor's NB topic) used by sensors which are no longer in the list, and initiates resources for new sensors. Then SG notifies the client for all changes made.

4.1.3.4 Detailed Description

In this section, message flow of various GB operations will be discussed at the Class level using unified modeling language (UML) collaboration diagrams.

4.1.3.4.1 Starting a Domain

The following **Figure 17** diagram shows the events happening when a domain is started.

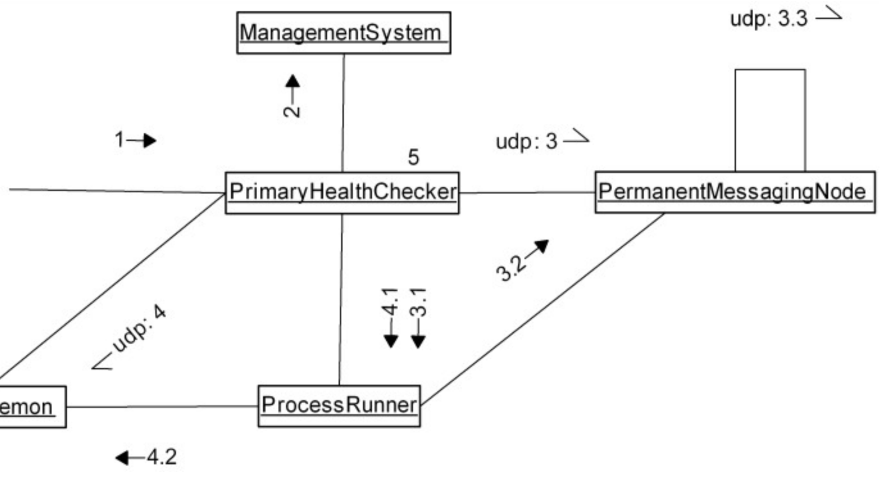


Figure 17. Event flow when starting a sensor grid domain

1. A user starts the domain by executing "runPrimaryHealthCheck.bat"

2. ManagementSystem.BootStrap() is called to initialize all system properties, environment variables and various user-defined properties from configuration files

3. Send a PingRequestMessage to the expected locator(s) of messaging node(s) registered in configuration files. If any messaging node does not respond with PingResponseMessage within 5 seconds, go to 3.1. Otherwise go to 4

3.1. For each messaging node not responding, send a request to ProcessRunner to start a PermanentMessagingNode process

3.2. ProcessRunner starts the messaging node process

3.3. Spawns a thread which continuously monitors the presence of itself by using UDP messages (ping request and response). Starts a BrokerNode (NB) according the configuration provided by configuration file (defaultMessagingNode.conf)

4. Send a PingRequestMessage to the expected locator(s) of ForkDaemon(s) registered in configuration files. If any ForkDaemon does not respond with PingResponseMessage within 5 seconds, go to 4.1. Otherwise go to 5

4.1. For each ForkDaemon not responding, send a request to ProcessRunner to start a ForkDaemon process

4.2. ProcessRunner starts the ForkDaemon process

5. PrimaryHealthChecker sleeps for 10 seconds to allow any pending processes to instantiate. Then it checks whether all messaging nodes and ForkDaemons are up and running. If yes, it sleeps for 30 seconds. Afterwards, it goes to step 3 and checks everything again

4.1.3.4.2 Starting BootstrapService of a Domain

When a domain is started, it undergoes the Bootstrap sequence shown in **Figure 18**.

1. Initialize the Bootstrap node from config file, including domain hierarchy and locators of ForkDaemons, RegistryForkDaemon, MessagingNodeDaemons. NB transport is initialized for NB communications with other domains

2. If the current domain is not a leaf node, register all sub-domains locally

3. If the current domain is not the root node, runs a thread that periodically sends a RegisterRenewMessage to the BootstrapService of its parent telling this domain's BootstrapService is running. If the domain is a leaf node, go to 3.1. Else go to 4

3.1. Starts a thread that periodically spawns a SystemHealthCheck process for each registered ForkDaemon.

3.2. Spawns a SystemHealthChecker process by sending a ForkProcessMessage to ForkDaemon with the "healthcheck" parameter

3.3. ForkDaemon spawns the Manager process with the "healthcheck" parameter.

3.4. Manager starts the SystemHealthChecker thread. System undergoes Normal Health Check Sequence (Please refer to Section 4.1.3.4.3 f o r details). BootstrapService waits 10 seconds for the reply from SystemHealthChecker

3.5. The replied status from SystemHealthChecker is either COMPLETE, UNKNOWN or RUNNING. Repeat 3.1 after some sleep

4. If the node is not a leaf node, spawn a thread that periodically checks the status of ALL RegisteredSubDomains (RSD). Under the Health Check mechanism, all RegisteredSubDomains are supposed to send a RegisterRenewMessage to its parent.

5. If no RegisteredRenewMessage is received from a SubDomain within a specified amount of time, the thread spawns a BootstrapService of the SubDomain remotely by sending a ForkProcessMessage to its ForkDaemon

6. ForkDaemon creates the BootstrapService of the SubDomain

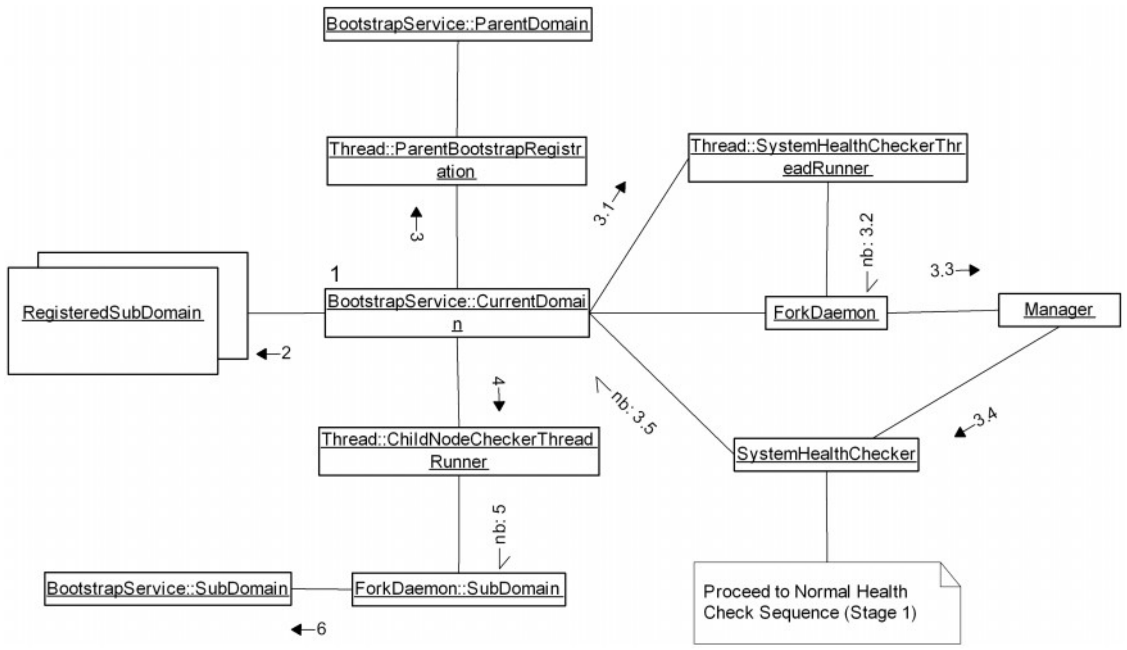

BootstrapService::ParentDomain
Thread::ParentBootstrapRegistration
Thread::SystemHealthCheckerThreadRunner
3.1
3
nb: 3.2
3.3
1
RegisteredSubDomain
BootstrapService::CurrentDomain
ForkDaemon
Manager
2
4
nb: 3.5
3.4
Thread::ChildNodeCheckerThreadRunner
SystemHealthChecker
nb: 5
BootstrapService::SubDomain
ForkDaemon::SubDomain
6
Proceed to Normal Health Check Sequence (Stage 1)

Figure 18. Starting BootstrapService of a Domain

4.1.3.4.3 Normal Health Check Sequence (Stage 1)

System Health Check has a number of stages. During the first stage (**Figure 19**), Bootstrap Service checks if the Registry is present. If not, creates a Registry process using the Fork Daemon.

1. After NB transport is initialized, a thread is started that automatically kills the the health checker if it is still running after 60 seconds

2. A thread is started that automatically notifies the BootstrapService at an interval of 2 seconds that the health checker is running

3. Checks if there is a Registry running in the domain by sending a RegistryQueryMessage to the defined Registry locator. If a RegistryQueryResponse message is received, go to 4. If no, go to 3.1

> 3.1. Try spawning a Registry process by sending a ForkProcessMessage to ForkDaemon. Max retries = 5. After each retry, repeat 3. If number of retries reached, health checker terminates with abnormal exit status
>
> 3.2. ForkDaemon creates the Registry process. Registry checks if persistent storage is used in configuration file (mgmtSystem.conf). If yes, go to 3.2.1. Otherwise persistent storage won't be used and everything will be saved in memory. Proceed to 3.3
>
> 3.2.1. Registry asks PersistantStoreFactory for an instance of WSContextStore, which is responsible for storing and retrieving settings from persistent storage (e.g., relational database)
>
> 3.2.2. WSContextStore is initialized by making connections to various components defined in WSContext and removing all previous entries (e.g., registered service adapters, service policy, service status etc.). If any errors occur during initialization, go to 3.3 and everything will be saved in memory
>
> 3.2.3. Registry loads all settings from WSContextStore to in memory hash tables

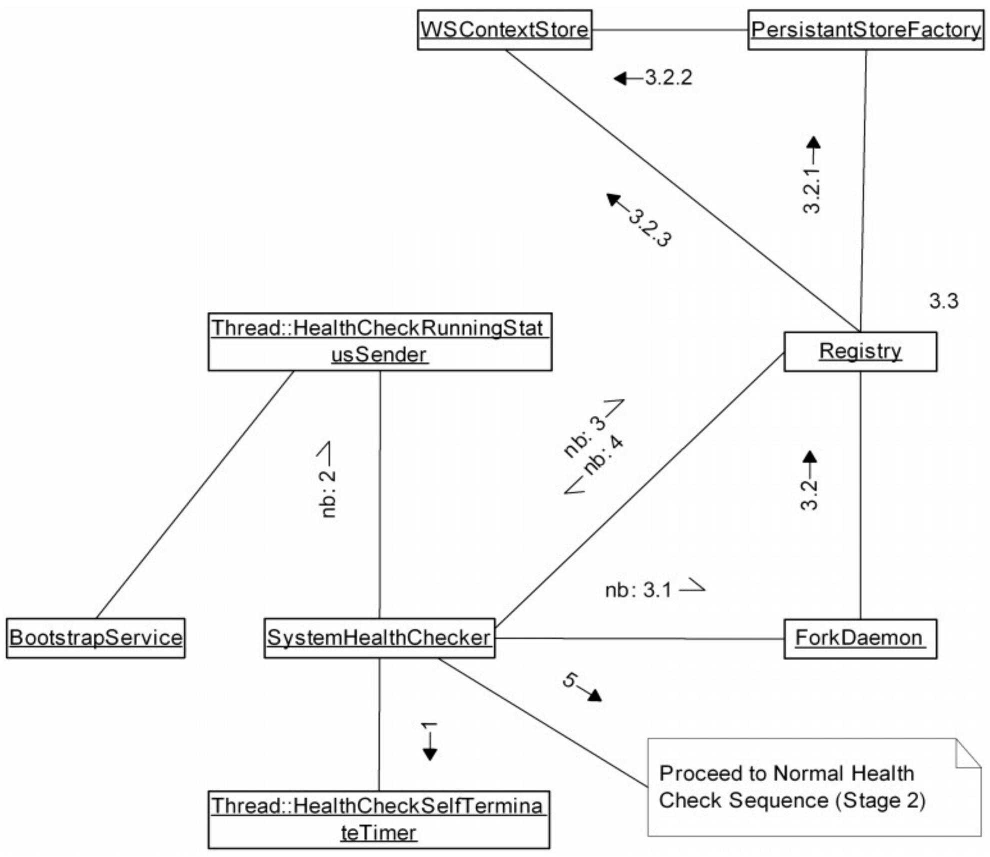


Figure 19. Normal Health Check Sequence (Stage 1)

3.3. Registry initializes NB transport by subscribing to two topics – one common to all registries and one uniquely identify itself. Registry spawning process has been finished. Go back to 3

4. Registry responds to SystemHealthChecker with the number of managers and service adapters expected in the domain.

5. System now enters health check stage 2.

4.1.3.4.4 Normal Health Check Sequence (Stage 2)

During the second stage of the System Health Check (show in

Figure 20), Bootstrap Service checks if enough Managers are spawned as defined in the configuration file.

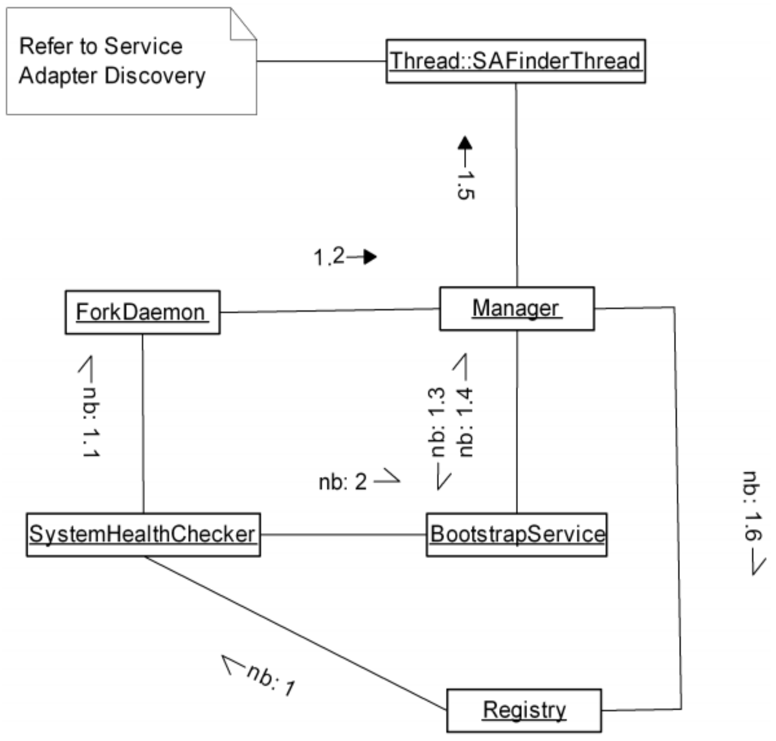


Figure 20. Normal Health Check Sequence (Stage 2)

1. The Registry responds to SystemHealthChecker with the number of managers and service adapters expected in the domain. If there are enough managers for all RegisteredServiceAdapters, go to 2. Otherwise go to 1.1

   1.1. For each Manager lacking, create a Manager process without the "healthcheck" parameter sending a ForkProcessMessage to ForkDaemon

   1.2. ForkDaemon creates the Manager process

   1.3. Request system configuration from BootstrapService, including locator of Registry, ForkDaemon

   1.4. BootstrapService replies with system configuration

   1.5. Initialize NB transport support. Starts a SAFinderThread which keep sending FindSAToManageMessage to Registry requesting corresponding ServiceAdapters to manage. If no reply from Registry, the request is repeated periodically at 2 second intervals. For details of this part, please refer to Section 4.1.3.4.6.

1.6. The Manager periodically sends a RegisterRenewMessage to the Registry to notify its presence

2. SystemHealthChecker sleeps for 10 seconds to allow any pending processes to instantiate. Then it checks whether all expected processes are up and running. If yes, it sends a SystemHealthCheck message to BootstrapService, notifying that System Health Check is completed and then terminates itself. Otherwise, it checks the system's health from stage one again (Section 4.1.3.4.3) and tries spawning the missing process(es).

4.1.3.4.5 Registered Service Adapter Health Check Sequence

SAMModule notifies the Service Adapter which Manager (**Figure 21**) that it should send heart beat messages to. The following sequence is followed by the Registered Service Adapter (RSA).

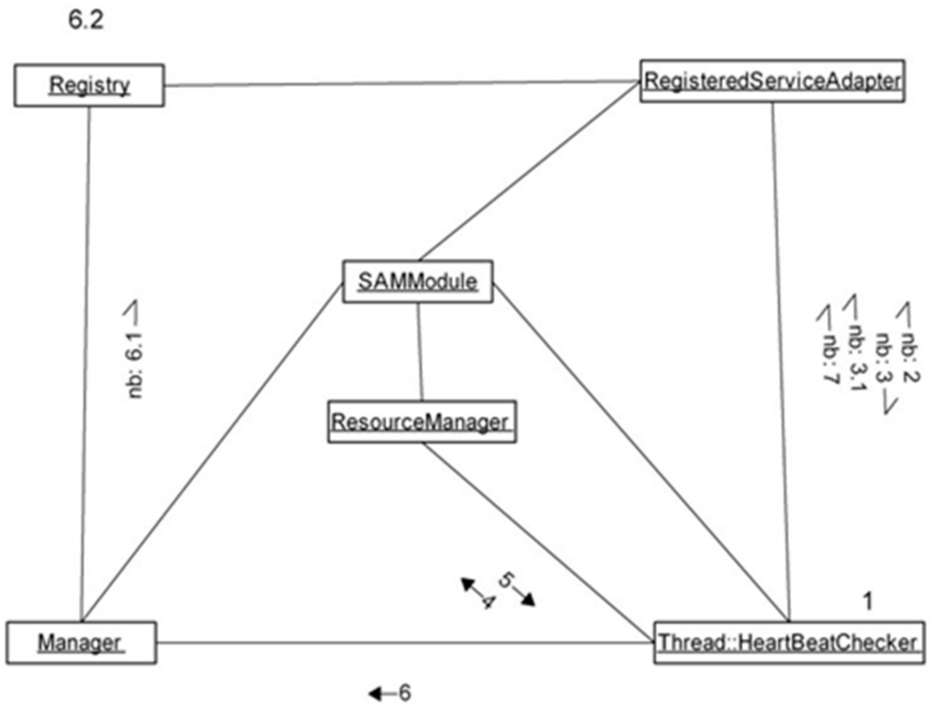


Figure 21. RSA Health Check Sequence

1. Checks if the associated RSA has sent a HEARTBEAT within the specified interval. If yes, sleep for a while and do 1 again. Else go to 2

2. Sends a GetCurrentManager message to the associated RSA to check if it is the RSA's current owner. If RSA replies, go to 3. Else go to 4

3. If UUID of RSA's current owner matches with this SAMModule, go to 3.1. Else go to 4.

    3.1. Sends a HEARTBEAT message to the RSA and wait. If RSA replies within a time limit, sleep for a while and do 1 again. Else go to 4

4. Ask ResourceManager(RM) whether to release the RSA.
5. If RM knows that the RSA is up and running, go to 7. Else go to 6
6. Notifies the Manager that the associated RSA is unreachable.
    6.1. Sends a UPDATE_SA_STATUS message to the Registry, saying that the RSA is UNREACHABLE
    6.2. Registry performs status update
7. Re-register with the RSA by sending a HEARTBEAT to it. Sleep for a while and do 1 again

4.1.3.4.6 Service Adapter Discovery

System Health Check (see **Figure 22**) checks if every Service Adapter is associated with its Manager.

1. SAFinderThread sends a FindSAToManageMessage to Registry. If persistent storage is used in the Registry, go to 1.1. Otherwise go to 1.2.
    1.1. Registry retrieves the information of a list of Registered Service Adapters from WSContextStore
    1.2. Registry replies with ServiceAdapterToManageMessage to the Manager if there is at least one SA which does not have an associated SAMModule. Status of the SA is set to MANAGED. At most one SA will be replied for each request. If there are no SA to manage, the Manager shutdowns itself.
2. For each SA, the Manager creates a SAMModule which manages the SA.
3. SAMModule creates a specific type of ResourceManager specified in the SA (in ServiceAdapterInfo), and starts the ResourceManager in a new Thread. For sensors, a SensorManager (ResourceManager for sensors) is instantiated

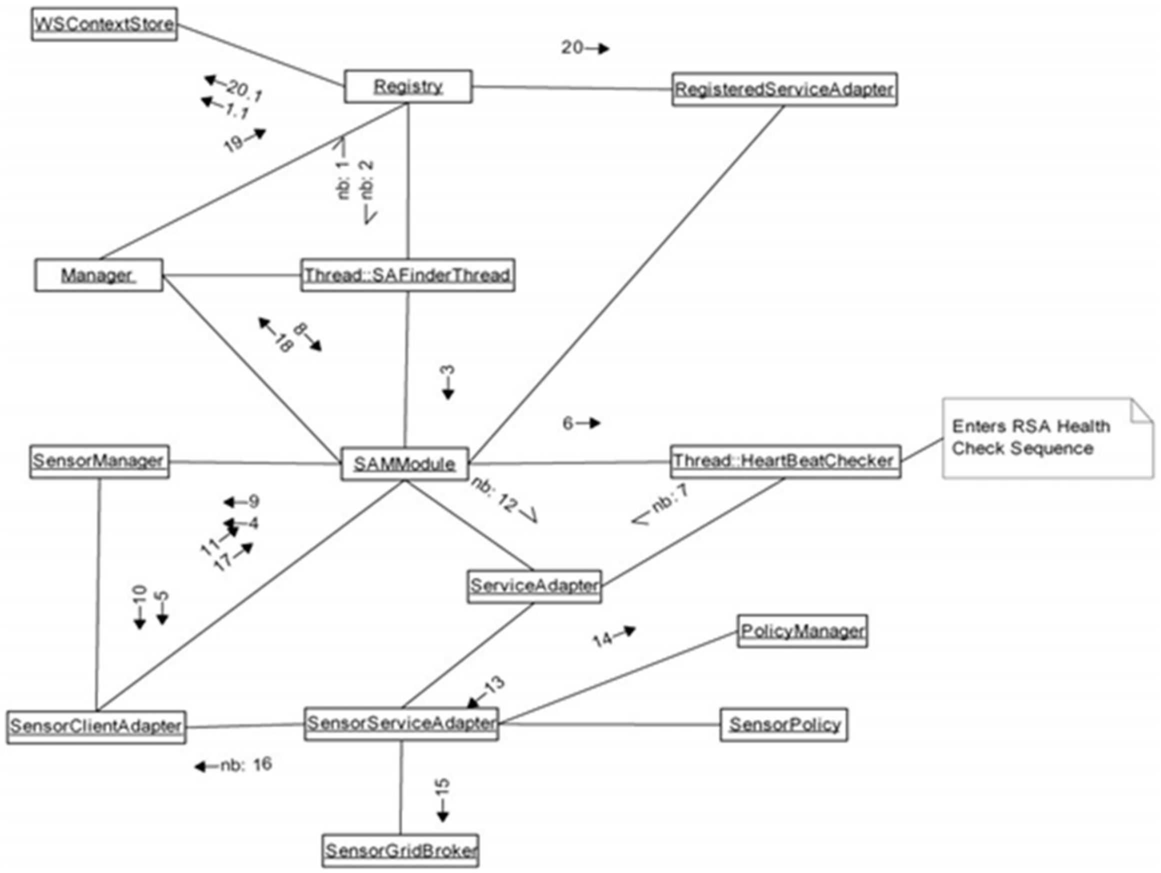


Figure 22. Message flow of service adapter discovery in a sensor grid

4. A SensorClientAdapter is instantiated. The SAMModule of SensorManager is passed as message sender and the locator of the associated SA is set as message destination

5. SAMModule starts a HeartBeatCheckerThread that periodically checks 1) if SA is up and running 2) if SA is still associated with this SAMModule (possibly taken control by other Managers)

6. Sends a setHeartBeatLocator message to SA to associate the SA with this SAMModule and tells SA the locator of Manager which heart beat messages should be sent to. Afterwards, HeartBeatCheckerThread enters the loop of SA health check (please refer to Section 4.1.3.4.5 - Registered Service Adapter Health Check Sequence)

7. Sends a GetServicePolicyMessage to SAMModule, request for the policy of the associated resource (i.e., sensor)

8. Forwards the request to SensorManager by calling getServicePolicy()

9. Invokes the associated SensorClientAdapter's getServicePolicy()

10. Sends a Wxf_Get message to the associated SensorServiceAdapter through SAMModule

11. Wraps the message with ServiceSpecificMessage and forwards it to the associated ServiceAdapter

12. Invokes processSOAPMessage of the associated SSA

13. If SensorPolicy has been defined, serialize it with PolicyManager. Otherwise, just create an empty message

14. If this is the first time SSA is assigned to a Manager, starts a SensorGridBroker which notifies SG of its presence

15. Sends back a response message with the serialized policy (if any)

16. Forwards the response to SAMModule

17. Forwards the response to Manager

18. Forwards the response to Registry

19. Updates the policy of the SA to the corresponding RSA in Registry. If persistent storage is used, go to 19.1; otherwise, go to 19.2

    19.1. The RSA is stored in WSContextStore

    19.2. The RSA is stored in memory

4.1.3.5 Deploying and Disconnecting sensors

4.1.3.5.1 Deploying a GPS Sensor

The message flow of deploying any sensors in a sensor grid is similar. For illustrative purposes, the message flow of deploying a GPS sensor is shown in **Figure 23**.

1. User chooses a domain and clicks “deploy”

2. UserUI creates a DeployDialog

3. User defines the policies of the sensor and clicks “ok”. A ForkProcessMessage is sent to the Registry to spawn a sensor client program

4. The message is forwarded to BootstrapService

5. The message is forwarded to ForkDaemon

6. ForkDaemon starts the type of sensor client program according to policy defined. Suppose user needs a GPS sensor. ForkDaemon creates a GPSManager process

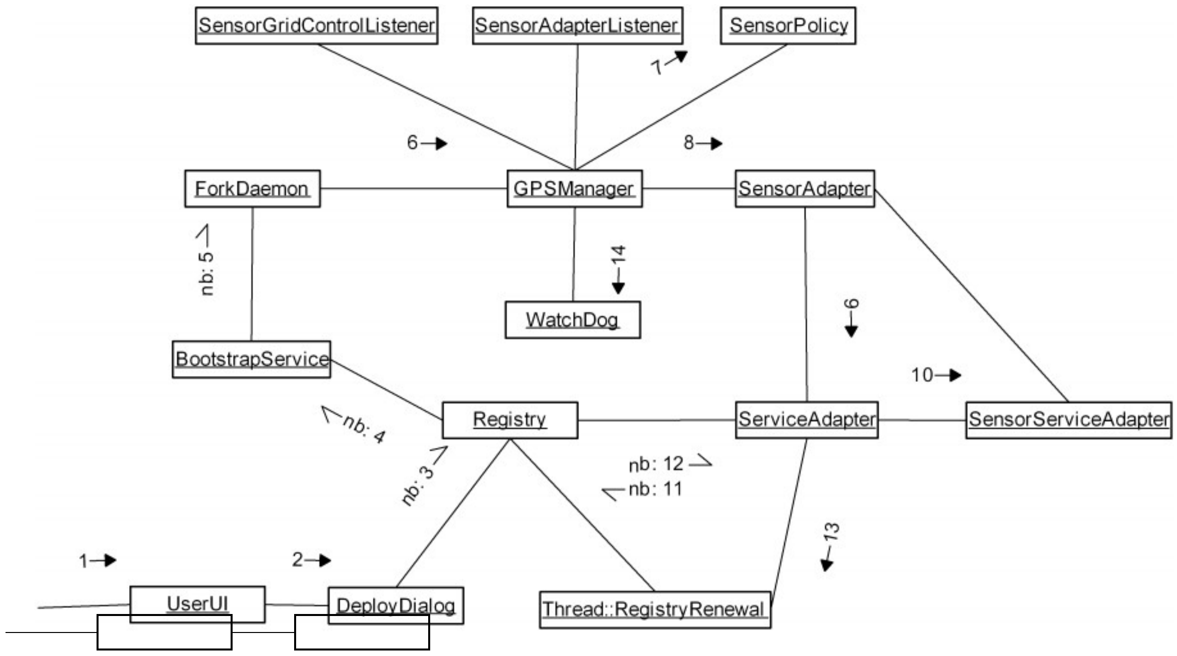


Figure 23. Deploying a GPS Sensor

7. Creates an instance of SensorPolicy according to the type of sensor and classification.

8. Creates an instance of SensorAdapter, passing in a SensorAdapterListener, SensorGridControlListener and SensorPolicy

9. Creates an instance of SA with parameters

"saType=cgl.hpsearch.sensor.SensorServiceAdapter" and

"manType=cgl.hpsearch.sensor.SensorManager"

10. Subscribes to the SA's own NB topic. Instantiates a SensorServiceAdapter according to

"saType"

11. Sends a RegisterRenewMessage to the Registry

12. If the SA is new to the Registry, it registers the SA, set SA's status to REGISTERED and replies to the SA with the new instanceId. If the SA is already registered, renew the status of SA according to its instanceId

13. Subscribes to a new NB topic according to the returned instanceId. Starts a new thread responsible for sending RegisterRenewMessage (heart beat) to the Registry. SA enters a state that keep tracking if NB connection is down. If yes, try to reconnect

14. GPSManager makes physical connection to the sensor, and starts a WatchDog which monitors the physical connection

After the new SA is registered in the registry, the Normal Health Check Sequence for Managers (Stage 2) will discover the new SA is not yet managed. A Manager will be assigned to it. For details please refer to section 4.1.3.4.4.

4.1.3.5.2 Disconnecting a Sensor

There are two ways to disconnect a sensor. The first way is to terminate the Sensor Client Program explicitly. The second way is to do it through GB's management console. The diagram provided at **Figure 24** shows the message flow of disconnecting a sensor through GB's management console.

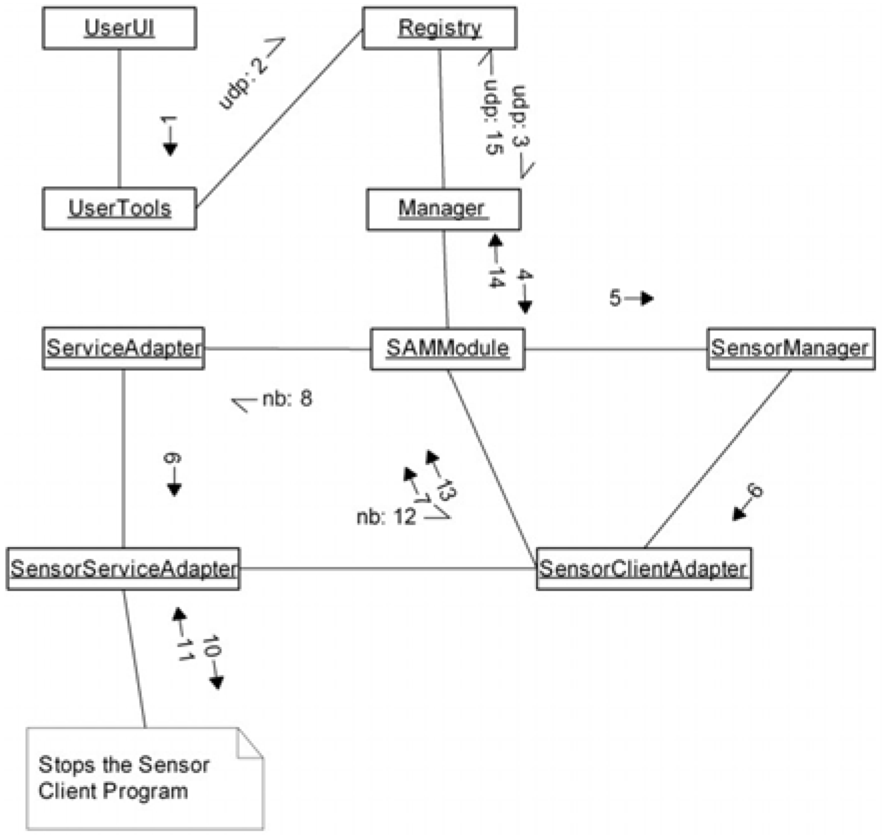


Figure 24. Disconnecting a sensor by using the Grid Builder management console

1. User selects a sensor in GB's management console and clicks "Stop." UserUI invokes sendRunMessage() of UserTools

2. UserTools creates a RunServiceMessage with parameters indicating the message is for disconnecting a sensor. The message is sent to Registry

3. Registry locates the Manager of the corresponding RegisteredServiceAdapter and forwards the message to it

4. Manager locates the corresponding SAMModule responsible for managing the ServiceAdapter and forwards the message to it

5. SAMModule forwards the message to the associated SensorManager

6. SensorManager forwards the message to the associated SensorClientAdapter

7. SensorClientAdapter sends a Wxf_Delete message to the associated SensorServiceAdapter through SAMModule

8. Wraps the message with ServiceSpecificMessage and forwards it to the associated ServiceAdapter

9. Invokes processSOAPMessage of the associated SSA

10. SensorServiceAdapter stops the sensor through SSAL.

11. An error report message is replied indicating if any error exists.

12. Forwards the reply to SensorClientAdapter.

13. Wraps the reply with a RunServiceResponse message, and sends it back to Registry through SAMModule.

14. Forwards the response to Manager.

15. Forwards the response to Registry.

16. Registry does not do anything to the response.

4.1.3.5.3 Deploying a Sensor using Container Service

Sensor Container Management Services

Prior to the implementation of Sensor Container Management Services (SCMS), every Sensor invoked used to live in its own Java Virtual Machine (JVM), hence there by consuming a lot of memory due to the overhead of individual “Run Times,” “Garbage Collectors,” etc. Due to limitation of system resources in a given Domain, the total number of Sensors that could be hosted/supported in a given Leaf Domain was limited by the degree of available system resources and not by the capacity of the underlining Broker. This was leading to Broker starvation. The SCMS implementation worked towards eliminating this issue. The new SCMS Architecture Diagram, presented in **Figure 25,** shows the architecture that was implemented to address this issue.

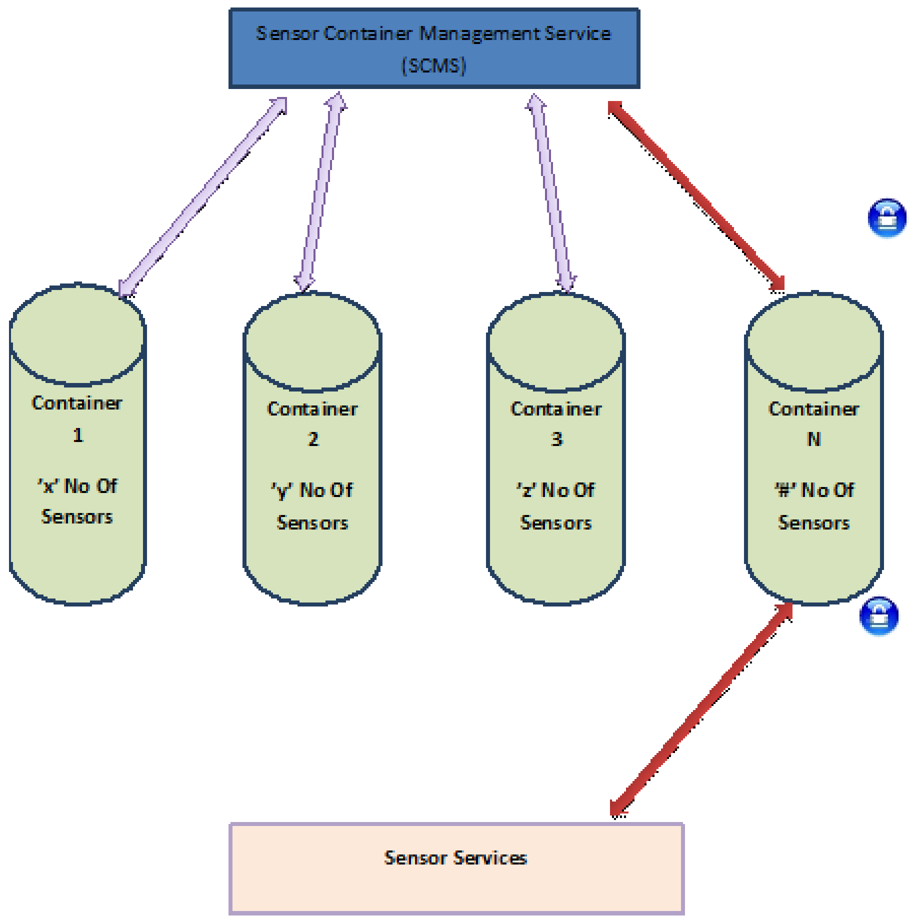

Sensor Container Management Service (SCMS)
Container 1 'x' No Of Sensors
Container 2 'y' No Of Sensors
Container 3 'z' No Of Sensors
Container N '#' No Of Sensors
Sensor Services

Figure 25. New SCMS Architecture Diagram

To support a huge number of Sensors in a single domain, the research team looked towards possible use of inter-process communication, and came up with the Container managed services approach. Using this approach, a container provides space for multiple sensors to live and service within a single JVM process and thereby share a single JVM resources. This results in huge decrease in the consumption of system resources. The Flow Diagram presented in **Figure 26** shows the inner working of SCMS. These are also mentioned below:

- SCMS is invoked using a script.
- SCMS brings up the first Container and provides the service Lock to the same.

- Once the container holding up the Service Lock is filled out, the same container shuts down its external services and releases the Lock.
- SCMS at that instance brings up another Container and provides the new container with the service lock.
- At any given instance if a few sensors have been shut down from a given container, the container registers to obtain the Service Lock and the same is handed over to the requesting Container by the SCMS once the current container holding the Service Lock is filled up and releases the Lock.
- SCMS also helps in cleaning up empty containers.

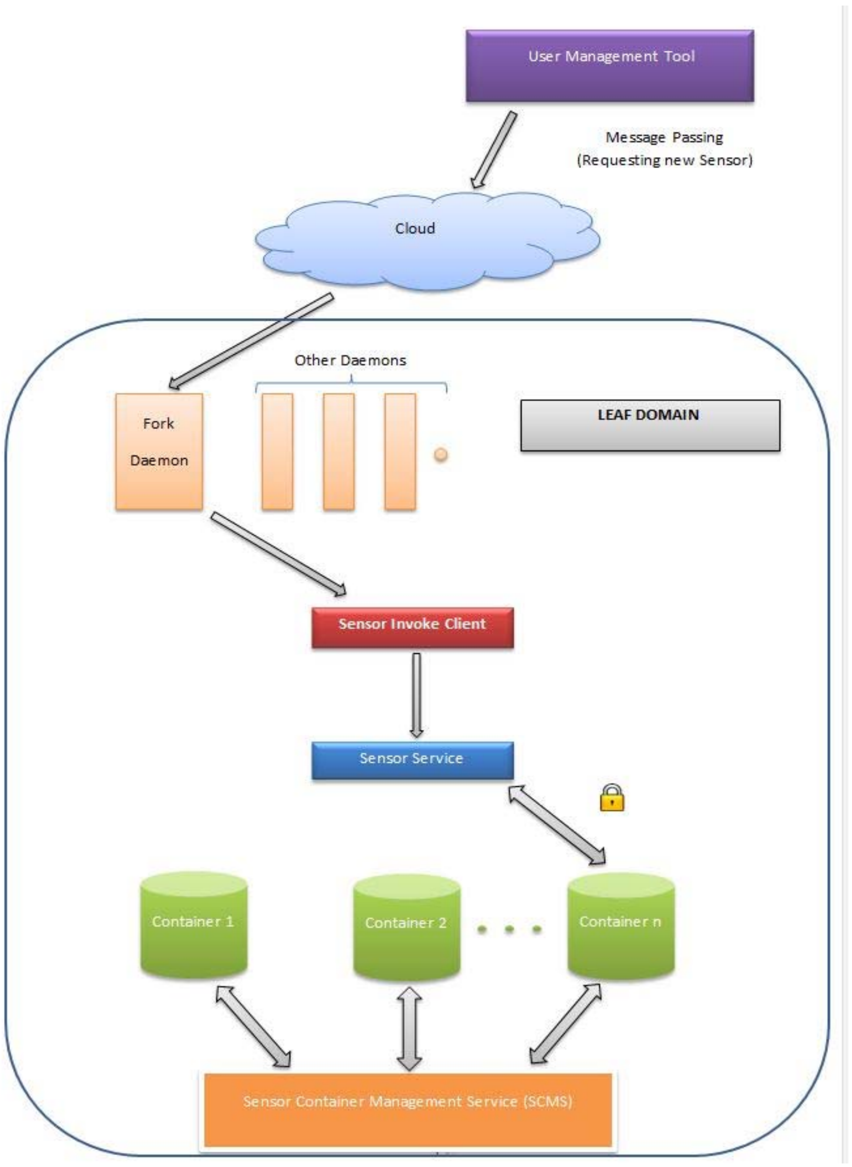


Figure 26. Architecture of the Container Service

A sensor invoke request is generated either directly by a script or by the User Management tool via ForkDeamon for a given Domain. In either case, it forks a new process containing the Sensor Invoke Client code. The Sensor Invoke Client repeatedly sends a request to the Sensor Service hosted by the Locked in Container until it receives a success response.

Once the Sensor Service receives the request it brings up a Sensor of a specific type requested within the container which holds the lock at that instance. Hence, multiple sensors could coexist within a single Container.

4.1.4 Sensor Grid Design Discussion

4.1.4.1 Overall Architecture of Sensor Grid and Related Modules

The SG is the brokering module of SCMW connecting the sensors, application clients and GB. The overall architecture for the SG is shown in

Figure 27. It serves two major functions: Message Brokering and Application Management. These are discussed next.

4.1.4.1.1 Message Brokering

Message Brokering enables the flow of messages among all parties including:

1. sensor data
2. sensor control messages
3. filtering requests and results
4. changes of sensor status
5. sensor policies

The following modules are essential for communication among the parties.

Application API

All kinds of applications communicate with SCMW through the same API. The Application API (shown at the top of

Figure 27) provides libraries for applications to:

1. access data and metadata of sensors
2. send control messages to sensors

3. notify change of sensor status
4. send filter requests to SCMW

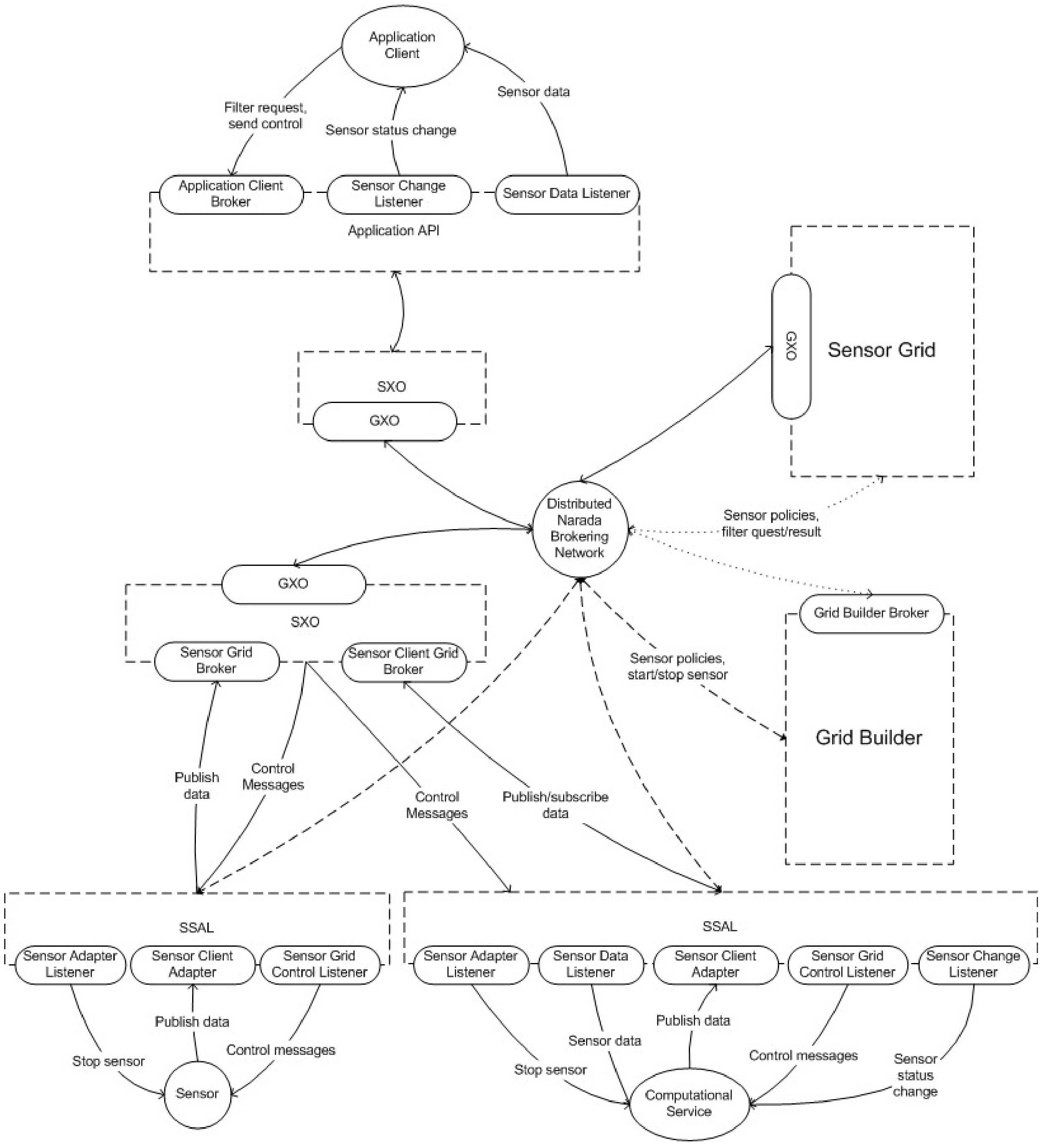


Figure 27. Overall Architecture of Sensor Grid and related Modules

These actions are done with the help of the following modules in the API:

Application Client Broker

Interface (shown as part of Application API in

Figure 27) used by application clients to send requests to SG, such as sending filter requests to SG and control messages to sensors (through SSAL).

Sensor Change Listener

Interface (shown as part of Application API in

Figure 27) used by application clients to receive messages from SG such as sensor status change.

Sensor Data Listener

Interface (shown as part of Application API in

Figure 27) used by application clients to receive data from sensors.

To support different applications, Application API in turn communicates with SCMW. For more detailed description of Application API, pleased refer to Section 4.1.2.5 above and section 4.1.5.2.

SSAL

All sensors communicate with SCMW through SSAL (shown at the bottom of **Figure 27**). Remember each sensor has a corresponding SCP to communicate with SCMW. SSAL provides libraries for sensors to do the following through SCP:

1. publish data.
2. receive control messages.
3. receive stop request from SCMW.
4. subscribe to data of another sensor.
5. listen to status change of subscribed sensor.

Not all kind of sensors have to use all the functions listed above. Remember, sensors can be further classified into normal sensors and Computational Service. In fact these two categories utilize different subset of classes in SSAL. Some of the important modules of SSAL are listed and discussed below. Note: These are shown from left to right at the bottom of **Figure 27**.

Sensor Adapter Listener

An interface for listening to stop requests from SCMW. The SCP should terminate upon receiving the request.

Sensor Data Listener

An interface for listening to data from subscribed sensors. Used by Computational Service.

Sensor Client Adapter

An interface for publishing data.

Sensor Grid Control Listener

The Sensor Grid Control Listener is an interface which sensors listen to for message control.

For more detailed description of SSAL please refer to Section 4.1.2.5 above.

Sensor Change Listener

The Sensor Change Listener is an interface for being notified when the subscribed sensor has any status change. Used by Computational Service.

4.1.4.1.2 Application Management

In SCMW, SG is responsible for maintaining the state of the whole system. For each deployed sensor and running application, SG caches down their presence and their relationships with one another. **Figure 28** below shows a scenario in which two applications and five sensors are connected to SG. The four tables show how SG maintains the state of the system, they include:

A list of online sensors (Table S)

SG maintains a list of online sensors which dynamically changes with the deployment status of the sensor.

Application to sensor mapping (Table A_S)

Each application needs a different set of online sensors according to its filtering criteria. This is to make sure that sensors which are not a concern of the application do not hold unnecessary resources. A table is maintained to remember this mapping.

Application to filter mapping (Table A_F)

Each application has its own filter, which are the criteria that define which sensors are needed by the application. The filter can be modified by the application at any time.

Sensor to sensor policy mapping (Table S_P)

Sensor Policies defines the characteristics of sensors. It is defined by GB before deployment. The sensor policy is obtained from GB and cached whenever a sensor is being deployed.

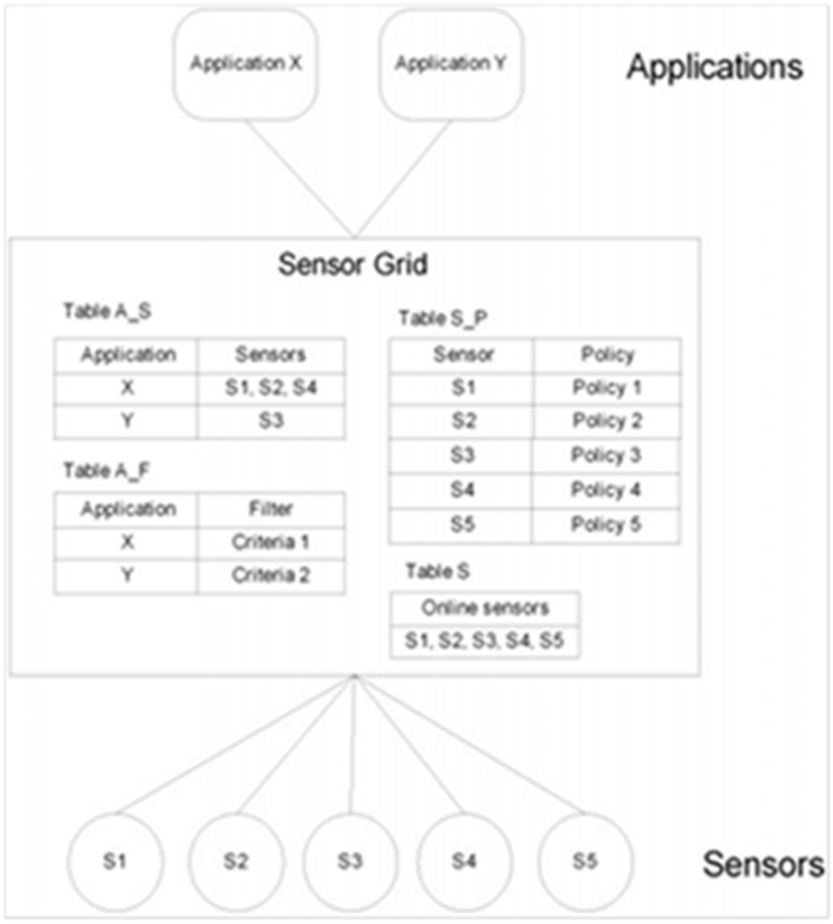


Table A_S

Application	Sensors
X	S1, S2, S4
Y	S3

Table A_F

Application	Filter
X	Criteria 1
Y	Criteria 2

Table S_P

Sensor	Policy
S1	Policy 1
S2	Policy 2
S3	Policy 3
S4	Policy 4
S5	Policy 5

Table S

Online sensors
S1, S2, S3, S4, S5

Figure 28. SG System Management

4.1.4.2 Significant Classes

4.1.4.2.1 Class Diagram

The class diagram for the sensor grid, sensor and application client is show in

Figure 29. The class descriptions for the classes shown in **Figure 29** are provided in the next section.

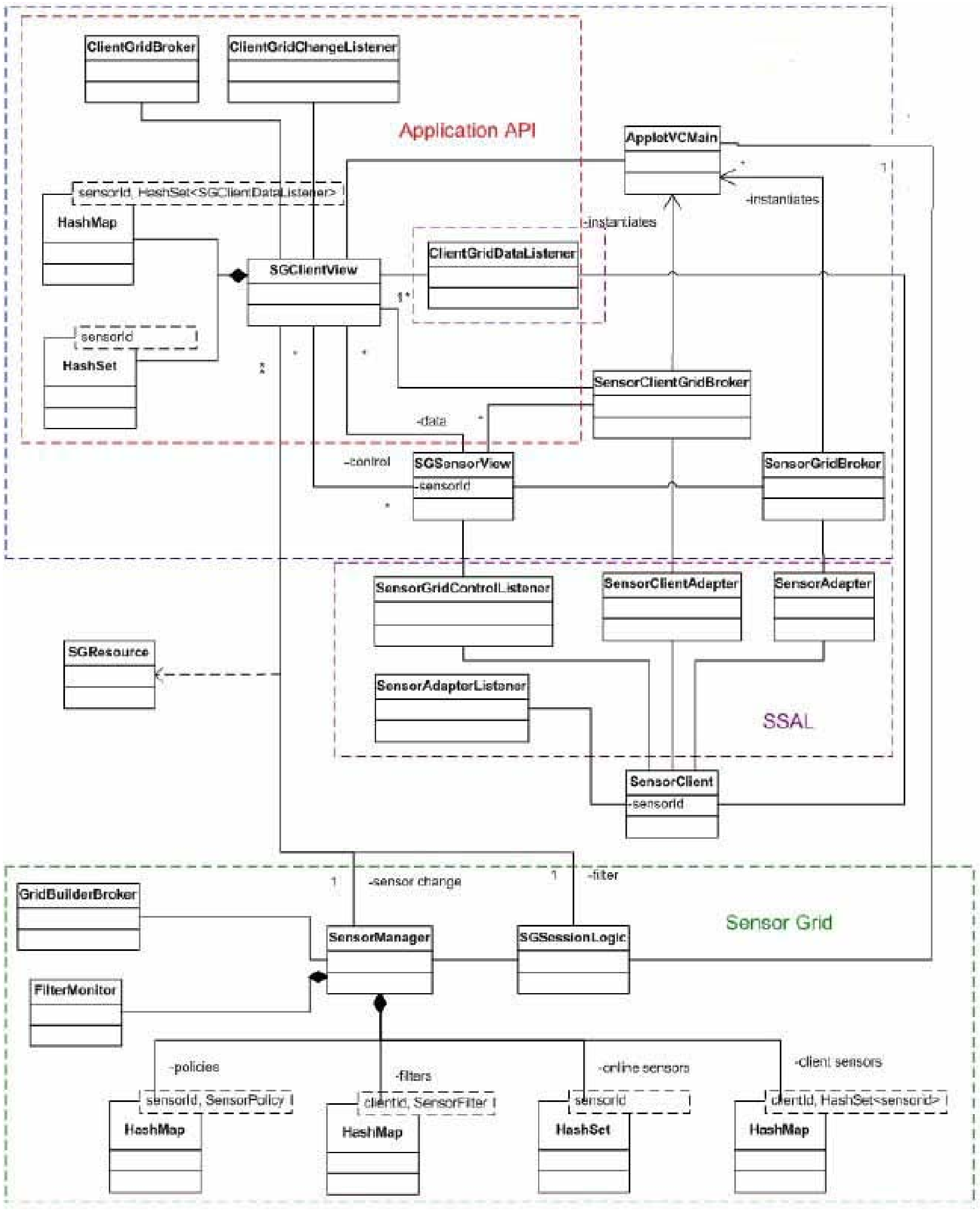


Figure 29. Class Diagram of SG, Sensor and Application Client

4.1.4.2.2 Class Description

This section provides brief description of important classes of SG and SSAL.

Class name:	ClientGridBroker
Package name:	com.anabas.sensorgrid.client
Description:	Part of the Application API. Provides the interface for external applications to communicate with SG and sensors.
Important interface:	setFilter(), sendControl(), subscribeSensorData(), unsubscribeSensorData()

Class name:	ClientGridChangeListener
Package name:	com.anabas.sensorgrid.client
Description:	Part of the Application API. Provides the interface for receiving sensor status change due to sensor deployment, disconnection and filtering
Important interface:	handleSensorInit(), handleSensorChange()

Class name:	SGClientView
Package name:	com.anabas.sensorgrid.session.sharedlet
Description:	Contains most of the application-client-side logic for the communication with SG and sensors, such as receiving sensor change, sending filter to SG and sending control messages to sensors. All NB topic and streams are handled here
Important interface:	setChangeListener(), startConnection(), subscribeSensorData(), unsubscribeSensorData(), setFilter(), sendControl()

Class name:	ClientGridDataListener
Package name:	com.anabas.sensorgrid.client
Description:	Part of the Application API, responsible for notifying the application on sensor data arrival. If the application clients wants to receive data from a particular sensor, it has to create a ClientGridDataListener for that sensor. Afterwards, the listener will be notified for data arrival
Important interface:	handleSensorData()

Class name:	SGSensorView
Package name:	com.anabas.sensorgrid.session.sharedlet
Description:	Contains most of the sensor-side logic for the communication with applications, such as publishing data and receiving control messages. All NB topics and streams are handled here
	setControlListener(), publishData()
Important interface:	

Class name:	SensorGridBroker
Package name:	com.anabas.sensorgrid.sensor
Description:	Brokers communication between SSAL, SG and sensors.
Important interface:	publishData(), close()

Class name:	SensorClientGridBroker
Package name:	com.anabas.sensorgrid.sensorclient
Description:	Brokers communication between SSAL, SG and service sensors.
Important interface:	publishData(), sendControl(), setFilter(), subscribeSensorData(), unsubscribeSensorData()

Class name:	SensorGridControlListener
Package name:	com.anabas.sensorgrid.sensor
Description:	Part of the SSAL. Provides the interface for receiving control messages
Important interface:	handleSensorControl()

Class name:	SensorAdapter
Package name:	com.anabas.sensor.sensoradapter
Description:	Part of SSAL. Provides the interface for sensors to publish data to applications
Important interface:	publishData(), start(), close()

Class name:	SensorAdapterListener
Package name:	com.anabas.sensor.sensoradapter
Description:	Part of SSAL. Responsible for receiving termination commands from GB
Important interface:	handleSensorConnectionLoss(), handleSensorStopRequest()

Class name:	FilterMonitor
Package name:	com.anabas.sensorgrid.session.sharedlet
Description:	Inner class of SensorManager responsible for periodic checkup to update the set of sensors for each application according to their corresponding filter
Important interface:	None

Class name:	SensorManager
Package name:	com.anabas.sensorgrid.session.sharedlet
Description:	Part of SG. Contains the logic for managing all connected applications and sensors. Maintains HashSets and HashMaps to cache sensor policies, applications' filters and sets of sensors mapped to each application.
Important interface:	addSensor(), removeSensor(), addClient(), startClient(), removeClient(), setFilter()

4.1.4.3 Important SG Features

4.1.4.3.1 NB Data Flow and Topic Management

Communication between applications, sensors and SG relies on NB for communication. This section provides a brief description of data flow between the three parties.

Each sensor creates a topic for publishing data and a topic for subscribing control messages. When an application is notified by SG for a new sensor, it subscribes to the two topics of the corresponding sensor directly for receiving data and publishing control messages.

For the communication between applications and SG, each application creates its own topic using its unique id for receiving sensor change notification. SG also creates a topic to receive filter requests from all applications.

The list of message streams and the related NB topic are shown in **Table 2**. The message flow between the SG, sensors and applications is shown in

Figure 30.

Table 2. NB Message Stream and Topics

Stream	**NB Topic**
T_SG	application/x-sharedlet-sensorgrid/private
T_CY	application/x-sharedlet-sensorgrid/client/CY
T_CX	application/x-sharedlet-sensorgrid/client/CX
T_S1_Data	application/x-sharedlet-sensorgrid/sensordata/S1
T_S1_Control	application/x-sharedlet-sensorgrid/sensorcontrol/S1
T_S2_Data	application/x-sharedlet-sensorgrid/sensordata/S2
T_S2_Control	application/x-sharedlet-sensorgrid/sensorcontrol/S2

4.1.4.4 Detailed Description

In this section, message flow of various SG operations will be discussed at the Class level using UML collaboration diagrams.

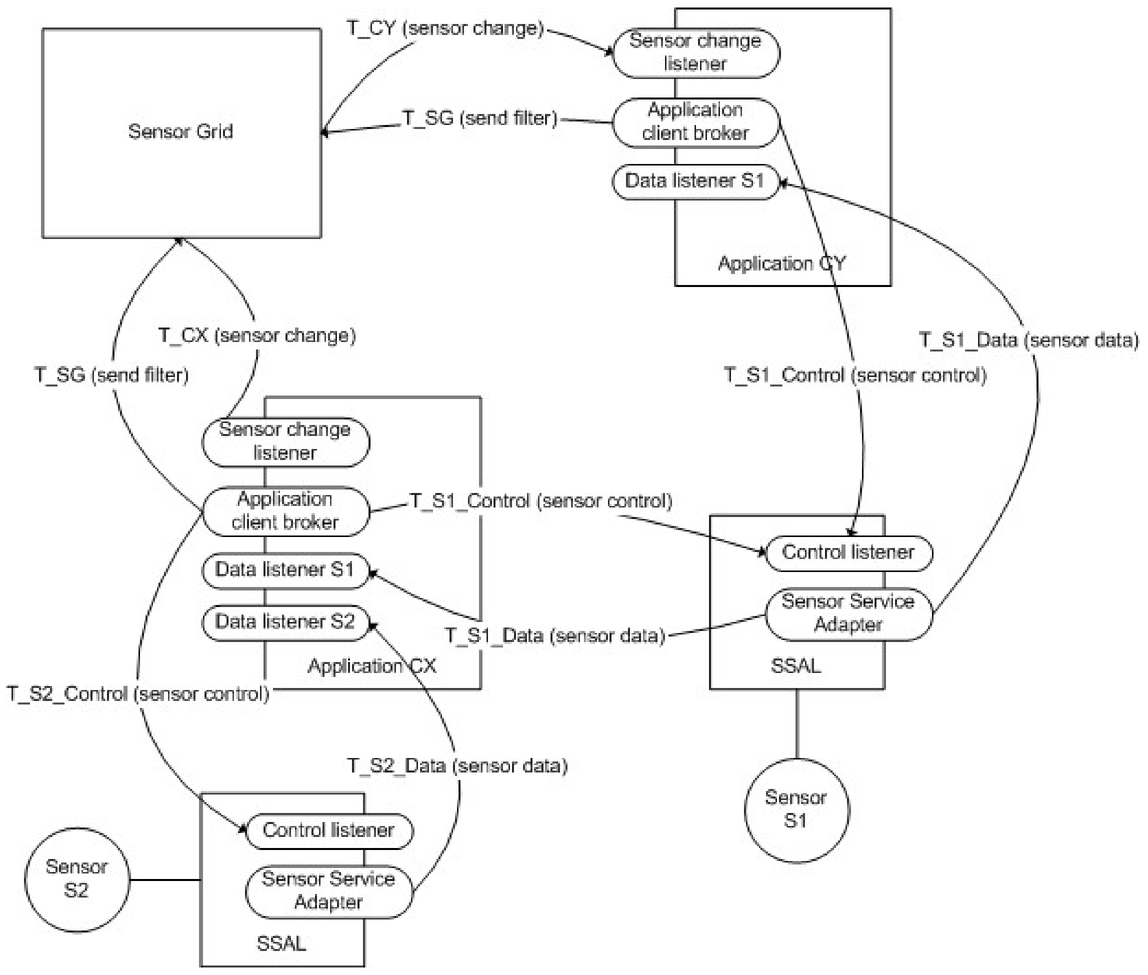


Figure 30. Message flow between a Sensor Grid (SG), applications and sensors

4.1.4.4.1 Sensor Grid Startup

The Sensor Grid startup sequence for a perpetual session is shown in **Figure 31Error! Not a valid bookmark self-reference.**. The descriptions of the steps for this startup sequence are:

1 An instance of SGSessionLogic is created by the framework

2 An instance of SensorManager is created, which is responsible for handling sensor-application interaction

3 An instance of GridBuilderBroker is created, which is responsible for obtaining SensorPolicy from GB

4 A thread is created to provide filtering for different application-clients every 5 seconds.

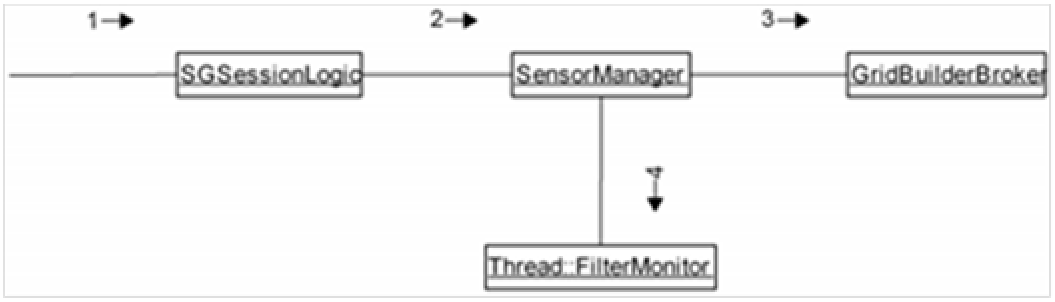


Figure 31. A Sensor Grid startup sequence

4.1.4.4.2 Deploying a Sensor

When deploying a sensor through the GB, a sequence of messages is invoked to enable the management of deployed sensors, as well as mechanisms to filter sensors based on sensor policies. The details of the message flow that occurs when a sensor is deployed through Grid Builder is illustrated in **Figure 32**.

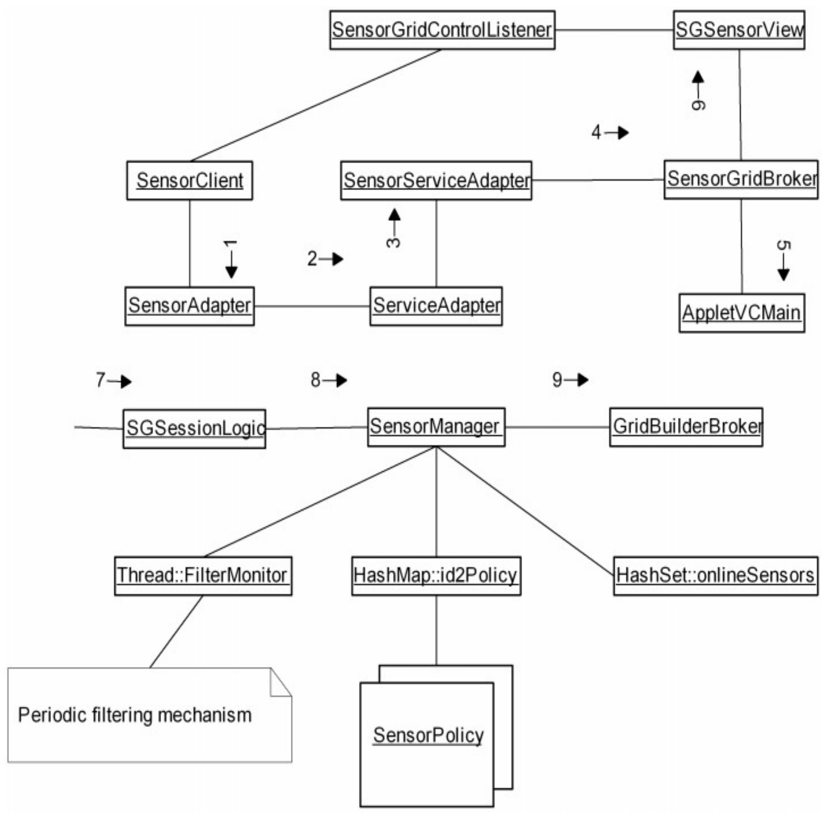


Figure 32. Message flow when deploying a sensor through the Grid Builder

A description of the sequence steps for this message flow is as follows:

1. The sensor client program instantiates SensorAdapter when it is started by GB
2. SA instantiates ServiceAdapter, which is later on managed by GB
3. SA instantiates SensorServiceAdapter, which resides in SSAL for communication with SensorManager of GB
4. SensorServiceAdapter instantiates SensorGridBroker, which communicates with SG
5. SensorGridBroker initializes all parameters needed for the sensor to join the Sensor Grid, including sensorId and system configuration, then instantiates AppletVCMain with all the parameters which tells the framework to prepare for a sensor client. It then sleeps for 5 seconds.
6. A SGSensorView is instantiated by the framework, which is responsible for message passing between application clients, sensors and SG. A unique NB stream is created for publishing sensor data and another one created for subscribing control messages. SensorGridBroker obtains a reference to SGSensorView from the framework and registers the SensorGridControlListener
7. The framework notifies that a new sensor has joined through the SessionListener interface of SGSessionLogic (userJoined()).
8. Invokes addSensor() of SensorManager. SensorManager caches the sensor in HashSet and its Policy in HashMap
9. Asks GB for SensorPolicy of the sensor through the GridBuilderBroker interface (getPolicy())
10. FilterMonitor Thread will notify all application-clients the presence of new sensor if it matches with the Filter.

4.1.4.4.3 Periodic Filtering

SG periodically checks the status of sensors and whether there are changes for each filter defined by applications. **Figure 33** shows the message flow associated with sensor filtering. The description of the message flow steps is as follows:

1. Every 5 seconds, the FilterMonitor Thread performs a filtering sequence. For each registered application-clients, the corresponding Filter object is obtained from a HashMap. Invokes doFiltering() of SensorManager.
2. Send a request to GB acquiring a list of sensors which matches the filtering criteria defined by the Filter.
3. GridBuilderBroker returns a list of sensors fulfilling the criteria.

4. Compare the list of returned sensors with the currently cached list of sensors for the application-client. Notifies the application-client all changes by sending a SENSOR_CHANGE message through a application-client specific NB stream.
5. Updates the cached list of online sensors in HashSet. Invokes handleSensorChange() of the registered ClientGridChangeListener (Sensor Change Listener).
6. ClientGridChangeListener notifies application client of sensor change. Application client performs corresponding actions.

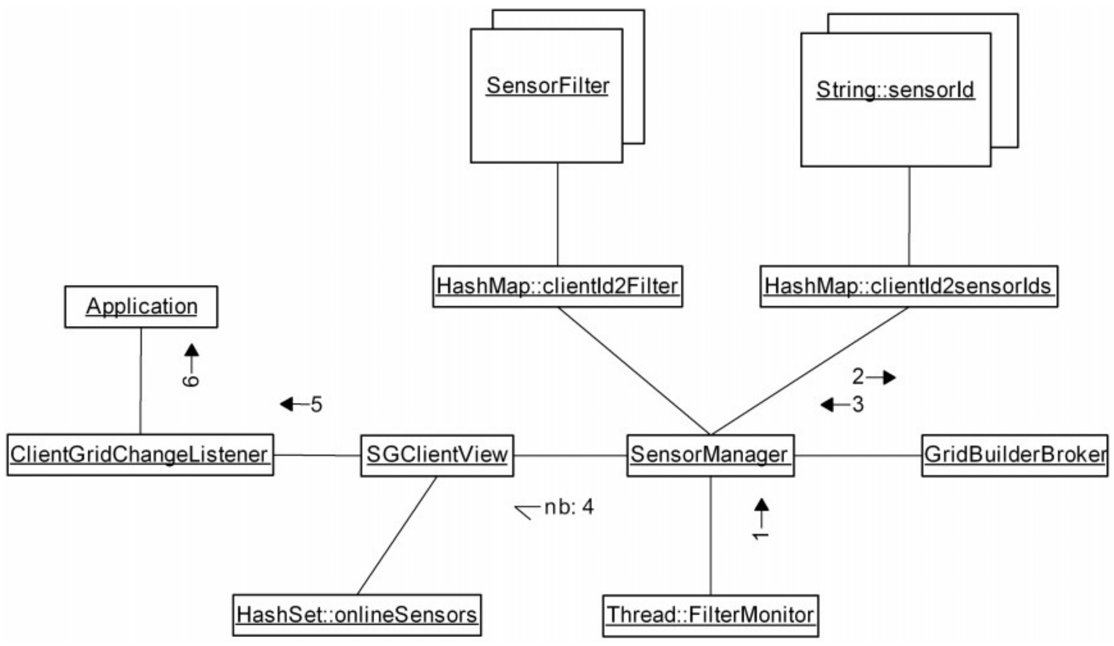


Figure 33. Sensor Grid message flow during periodic sensor filtering

4.1.4.4.4 Application Client Joining a Sensor Grid (SG)

When a sensor grid application client joins an SG, the message flow is as illustrated in **Figure 34**. Furthermore, a discussion of this message flow is as follows:

1. The application-client which implements the ClientGridChangeListener (Sensor Change Listener) interface, instantiates an instance of ClientGridBroker (Application Client Broker)
2. ClientGridBroker initializes all parameters needed for the application to join the Sensor Grid, including a generated client id which is unique to the system and client's system configuration, then instantiates AppletVCMain with all the parameters which tells the framework to prepare for an application client. Sleeps for five seconds.

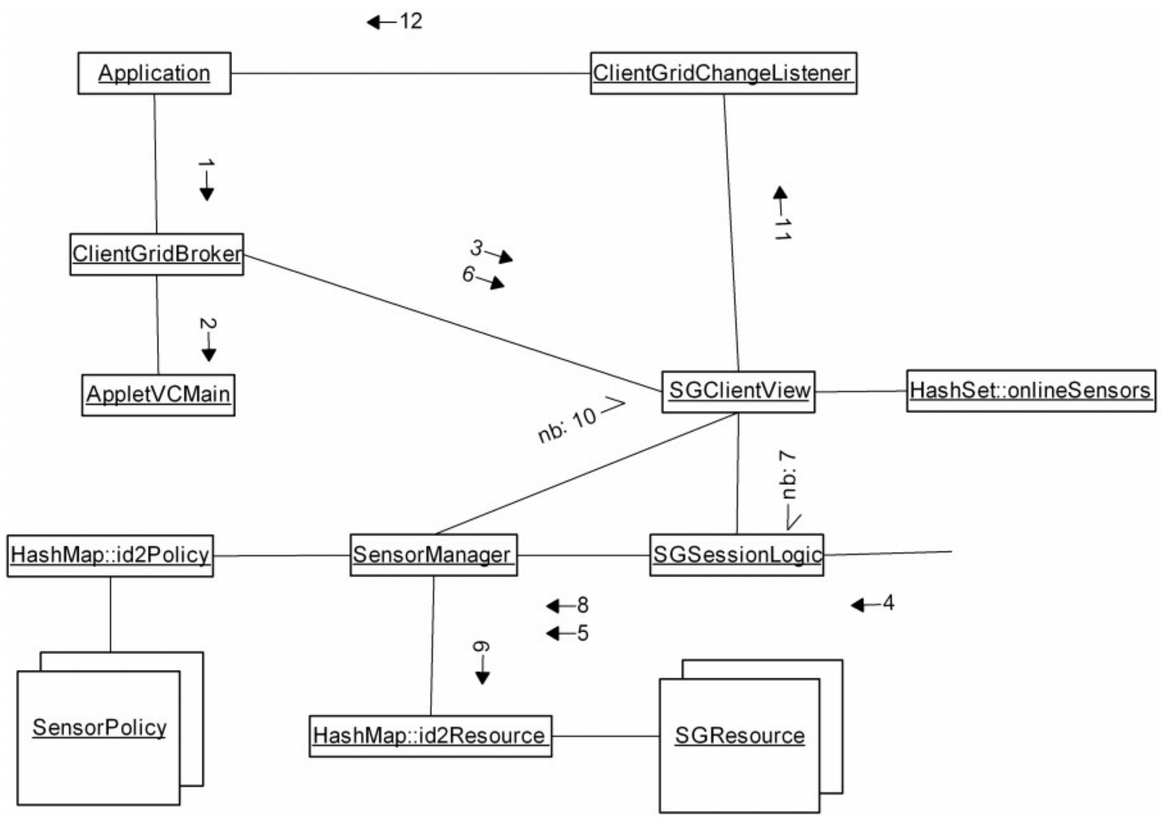


Figure 34. Message flow when an application joins a sensor grid

3. Updates the cached list of online sensors in HashSet. Invokes handleSensorInit() of the registered ClientGridChangeListener.
4. ClientGridChangeListener notifies application client of sensor change. Application client performs corresponding actions.
5. A SGClientView is instantiated by the framework, which is responsible for message passing between application clients, sensors and Sensor Grid. A unique NB stream is created for subscribing messages from Sensor Grid (e.g., sensor change information). ClientGridBroker obtains a reference to SGClientView from the framework and registers the ClientGridChangeListener.
6. The framework notifies that a new application client has joined through the SessionListener interface of SGSessionLogic (userJoined()).
7. Invokes addClient() of SensorManager. SensorManager initializes NB streams for communication with application client.
8. Registers application client's ClientGridChangeListener. Invokes SGClientView's startConnection().

9. Sends a START_CLIENT message with its client id.
10. Forwards the request to SensorManager.
11. Creates a HashMap which maps the id of all online sensors to SGResource instances wrapping the policy and status of the sensors.
12. Sends a INIT_SENSOR message to the client, containing the created HashMap.

4.1.4.4.5 Sensor Publishing Data

After a sensor is deployed in a sensor grid, a real-time stream of sensor data and metadata will be published to the sensor grid. The application clients which have subscribed to the live streams are notified of this data. The details of the message flow that occurs is illustrated in **Figure 35**.

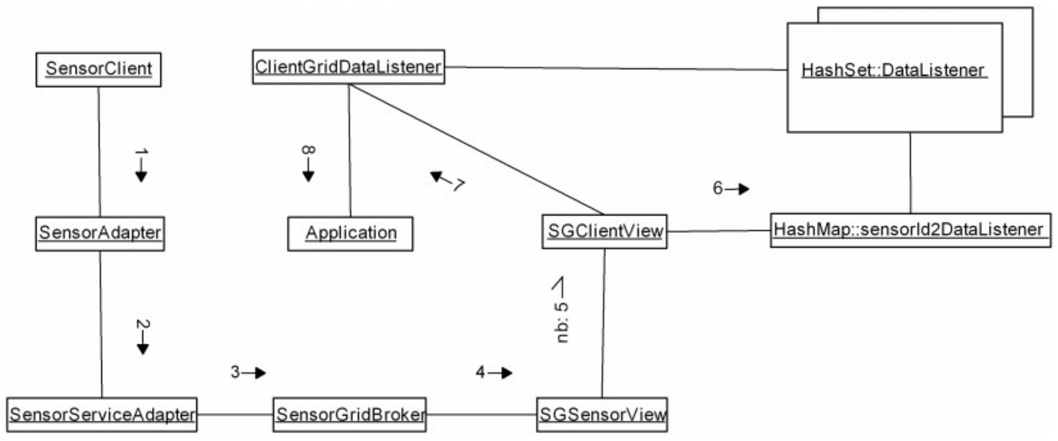


Figure 35. Message flow from deployed sensors to applications in a sensor grid

The description of the message flow steps is as follows:

1. SensorClient publishes data by calling publishData() of SensorAdapter
2. SensorAdapter forwards the data to SensorServiceAdapter by calling publishData()
3. SensorServiceAdapter forwards the data to SensorGridBroker by calling publishData()
4. The data is forwarded to SGSensorView
5. Broadcast the data through the unique NB stream for the sensor
6. For ALL the SGClientViews which have subscribed to data from this sensor, the registered ClientGridDataListeners (Sensor Data Listener) is located.
7. Each ClientGridDataListener found is notified of the data arrival by invoking handleSensorData().
8. Notifies the application for data arrival.

4.1.4.4.6 Subscribing Sensor Data

Applications that implement the SCMW API could receive relevant live sensor streams in the sensor grid by subscribing to them. The message flow that occurs when an application receives a live data stream of a deployed sensor for which it has subscribed is shown below in **Figure 36**.

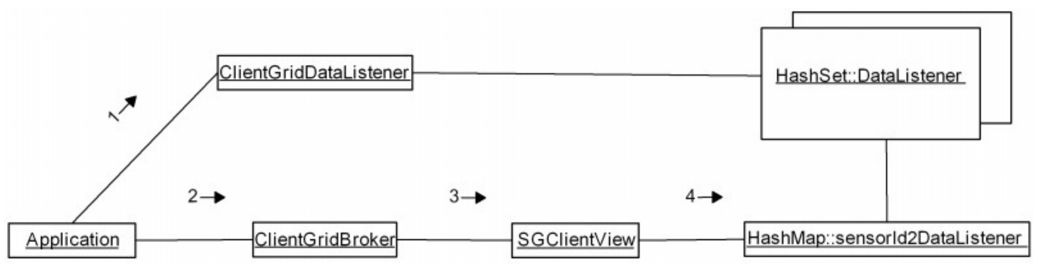


Figure 36. Message flow from a sensor grid to a subscribing application

The description of the message flow steps is as follows:

1. After application client knows the presence of a sensor, it creates an instance of ClientGridDataListener (Sensor Data Listener) for the sensor
2. Call subscribeSensorData() and provides the sensor id and ClientGridDataListener as parameter
3. Forwards the call to SGClientView
4. Register the ClientGridDataListener so that when sensor data arrives the listener will be notified. If this is the first request of subscribing data from this sensor, subscribes to the NB stream unique to the sensor

4.1.4.4.7 Setting a Filter

The design of SCMW supports filtering of sensor streams in a sensor grid to facilitate construction of UDOP for situational awareness. The message flow of an application setting up a filter query is shown in **Figure 37**. The description of the message flow steps is as follows:

1. Application client instantiates a SensorFilter object according to application-specific filter criteria.
2. Initiates a setFilter() request to ClientGridBroker, using the SensorFilter as parameter.
3. Forwards the request to SGClientView.
4. Sends a FILTER_MSG message to SG through NB, together with the SensorFilter object
5. Pass the SensorFilter object to SensorManager.

6. Send a request to GB acquiring a list of sensors which matches the filtering criteria defined by the Filter.
7. GridBuilderBroker returns a list of sensors fulfilling the criteria.
8. Compare the list of returned sensors with the currently cached list of sensors for the application-client. Notifies the application-client of all changes by sending a SENSOR_CHANGE message through a application-client specific NB stream.
9. Updates the cached list of online sensors in HashSet. Invokes handleSensorChange() of the registered ClientGridChangeListener (Sensor Change Listener).
10. ClientGridChangeListener notifies application client of sensor change. Application client performs corresponding actions.

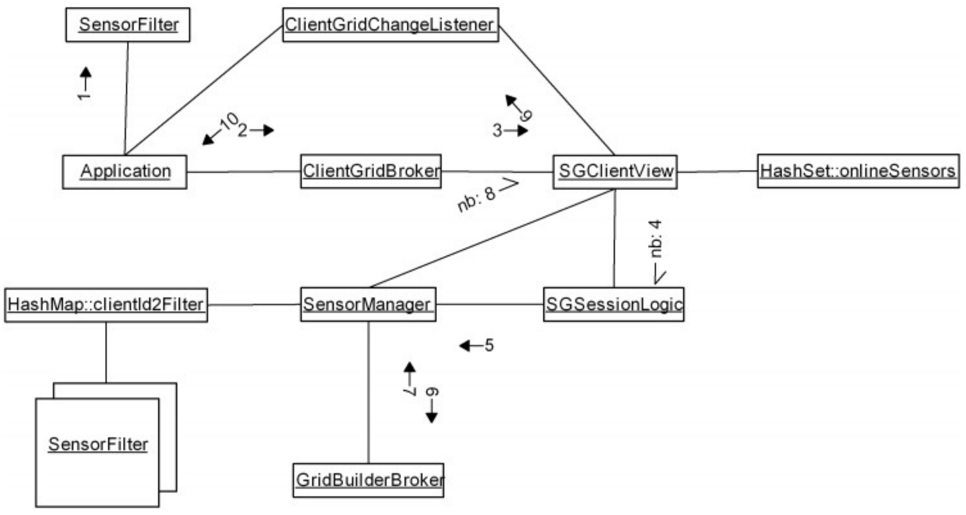


Figure 37. Message flow of filter setup in a sensor grid

4.1.4.4.8 Sending Control to a Sensor

Some sensors do not only send live streams to a sensor grid. They could receive control information from users or applications and respond with sensor information that corresponds to received control information. The message flow of an application sending a control message to a sensor is illustrated in **Figure 38**. The description of the message flow steps is as follows:

1. Application client invokes sendControl() of ClientGridBroker with the specified sensor id and control message recognizable by the sensor
2. Forwards the request to SGClientView
3. Sends the SENSOR_CONTROL to the sensor through a unique NB stream for the sensor
4. Forwards the control message to the registered SensorGridControlListener by handleSensorControl()

5. Notifies SensorClient that a control message is received. The sensor client performs the corresponding actions

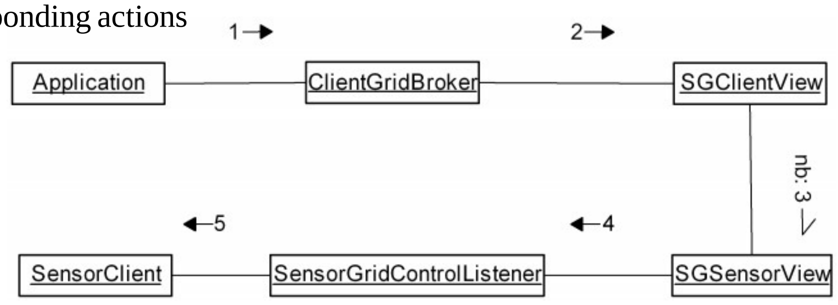


Figure 38. Message flow of control messages from applications to sensors in a sensor grid

4.1.4.4.9 Disconnecting a Sensor

To disconnect a sensor, one of the ways is to stop the sensor client program through GB's management console. The diagram shown in **Figure 39** presents the message flow of disconnecting a sensor this way.

The description of the message flow steps is as follows:

1. A disconnection request is received from GB (please refer to section 4.1.3.5.2 f o r details). The processWxfDelete() of SensorServiceAdapter is invoked.
2. Reports the running status of the associated sensor client program by sending a Wxf_DeleteResponse message to SensorServiceAdapter. If the sensor client program is running, go to 3. Otherwise, it does nothing and exits.
3. Invokes close() of SensorGridBroker.
4. Notifies the framework to dispose resource allocated to the sensor by calling allWindowsClosed() of AppletVCMain.
5. Notifies the associated SensorAdapterListener to terminate the sensor client program by calling handleSensorStopRequest().
6. SensorClient disconnect all connections and exits.
7. The framework notifies SGSessionLogic that the sensor has disconnected by invoking userLeft().
8. Invokes removeSensor() of SensorManager.
9. Removes the cached SensorPolicy and status for this sensor. For each application client, removes the sensor from the cached list of sensors associated with it, then notifies the application client by sending a SENSOR_CHANGE message through the unique NB stream for the client.

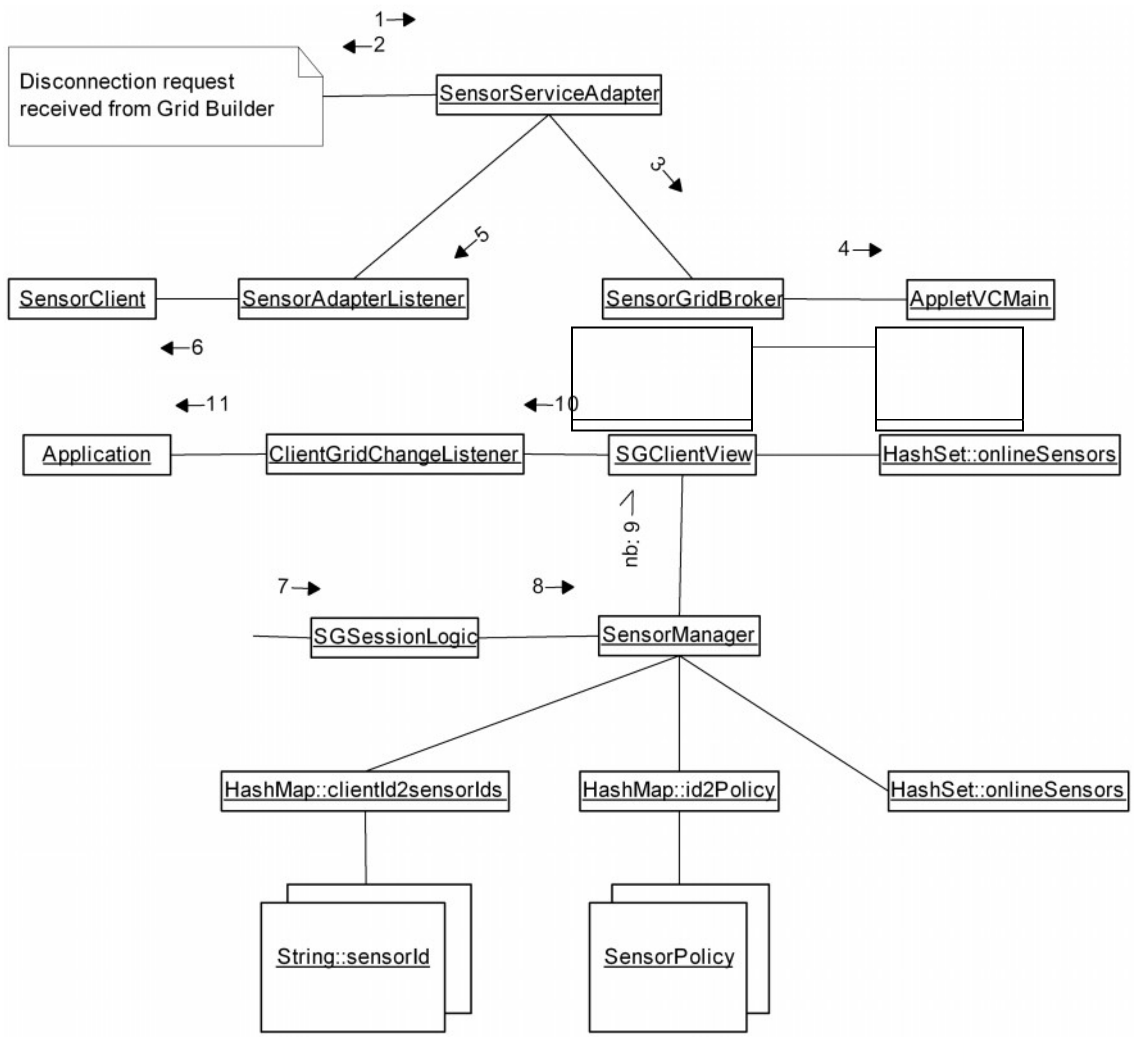


Figure 39. Message flow when disconnecting a deployed sensor from a sensor grid

10. Updates the cached list of online sensors in HashSet. Invokes handleSensorChange() of the registered ClientGridChangeListener (Sensor Change Listener).
11. ClientGridChangeListener notifies application client of sensor change. Application client performs corresponding actions.

4.1.5 SCMW Application Program Interface (API) and Sensor Service Abstraction Layer (SSAL)

4.1.5.1 Overview of the SCMW API

The SCMW Application Program Interface (API) allows any third party application to connect and utilize functions provided by SCMW. A graphical depiction of the API is shown in

Figure 40.

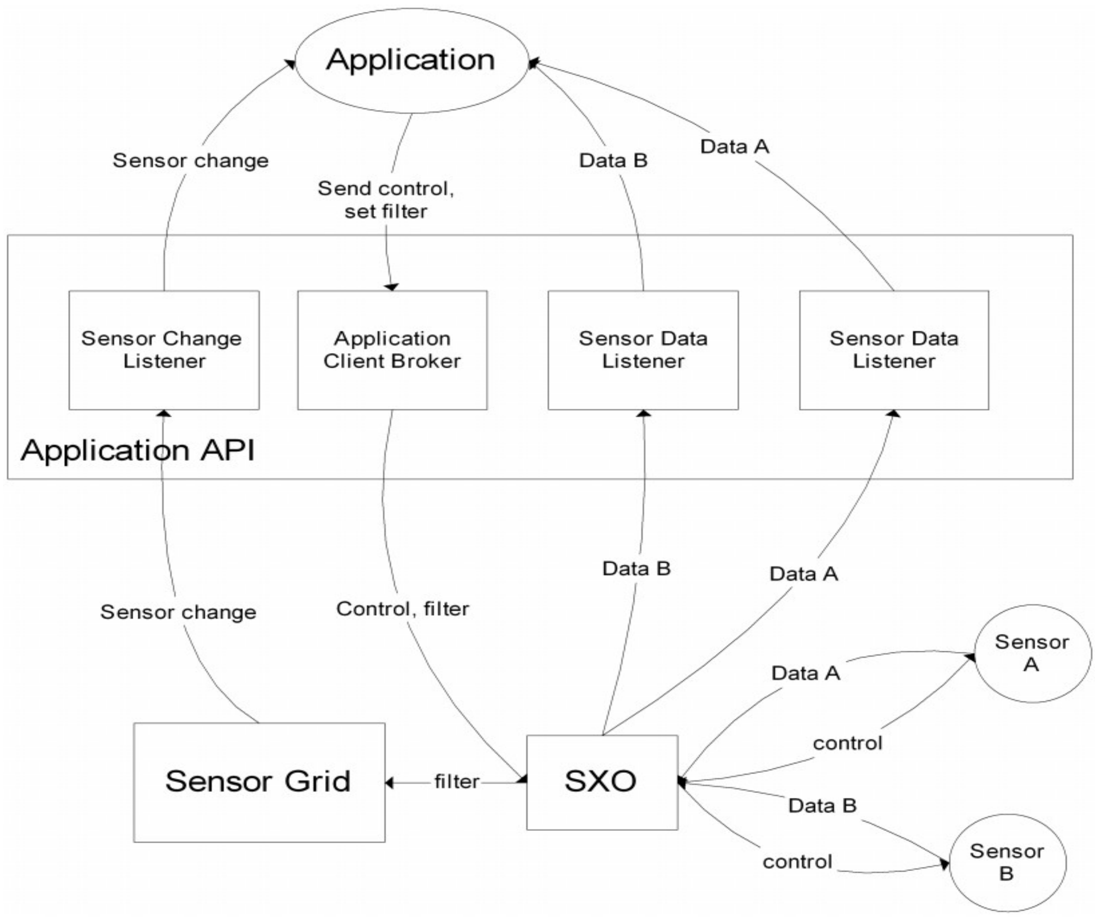


Figure 40. SCMW Application Programming Interface

An application can do the following through the SCMW API:

1. Obtain the policies and data of all sensors which are currently up and running
2. Selectively subscribe to sensors with their policies fulfilling filtering criteria defined by the application
3. Send control messages to sensors
4. Dynamically be notified for new sensors which fulfill the filtering criteria, and for sensors which have been disconnected

To use the SCMW API, an application has to instantiate an Application Client Broker (ClientGridBroker) and implement the Sensor Change Listener (ClientGridChangeListener) interface. Moreover, a Sensor Data Listener (ClientGridDataListener) has to be created for subscribing to the data stream of each sensor.

4.1.5.2 Sensor Service Abstraction Layer (SSAL)

4.1.5.2.1 Overall Sensor Service Abstraction Layer Architecture

The SSAL provides a common interface for all kinds of sensors. A high-level architecture view of the SSAL is shown in **Figure 41**. Sensor developers add new sensors to SCMW by writing an SCP which connects to SCMW through libraries in the SSAL.

Internally, SSAL communicates with GB for sensor management (e.g., creation, registration, definition) and SG for run-time management (e.g., data publishing, receiving control messages).

In SSAL, sensors are categorized into two categories:

Normal Sensors – Sensors which take input from external environment. The input data is external to SCMW.

Computational Service – Sensors which do not take input from the environment. Instead, they take output of other sensors as input, perform various computations on the data, and output the processed data finally

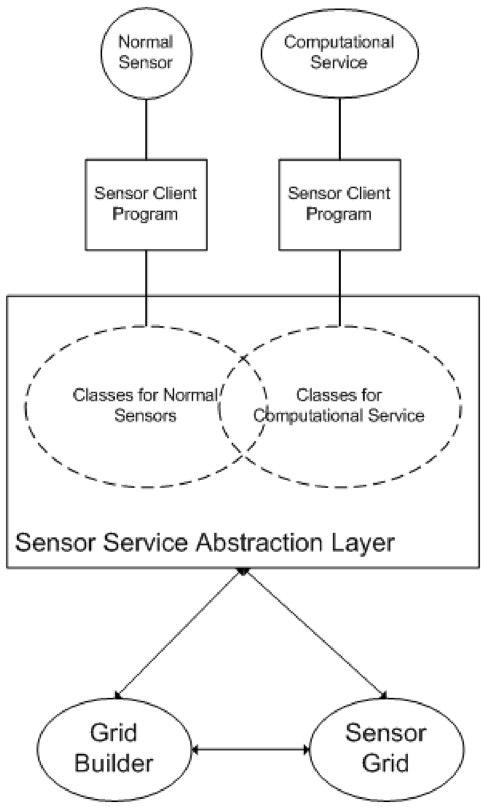


Figure 41. A high-level architecture of the Sensor Service Abstraction Layer (SSAL)

Functionally, the two different categories of sensors are supported by two different sets of classes in SSAL. Some classes are shared between the two categories for common function.

4.1.5.2.2 SSAL Architecture for General Sensor Services

Figure 42 shows the SSAL architecture for general sensors to be wrapped and deployed as sensor services. The following subsections explain the message flow for some basic operations.

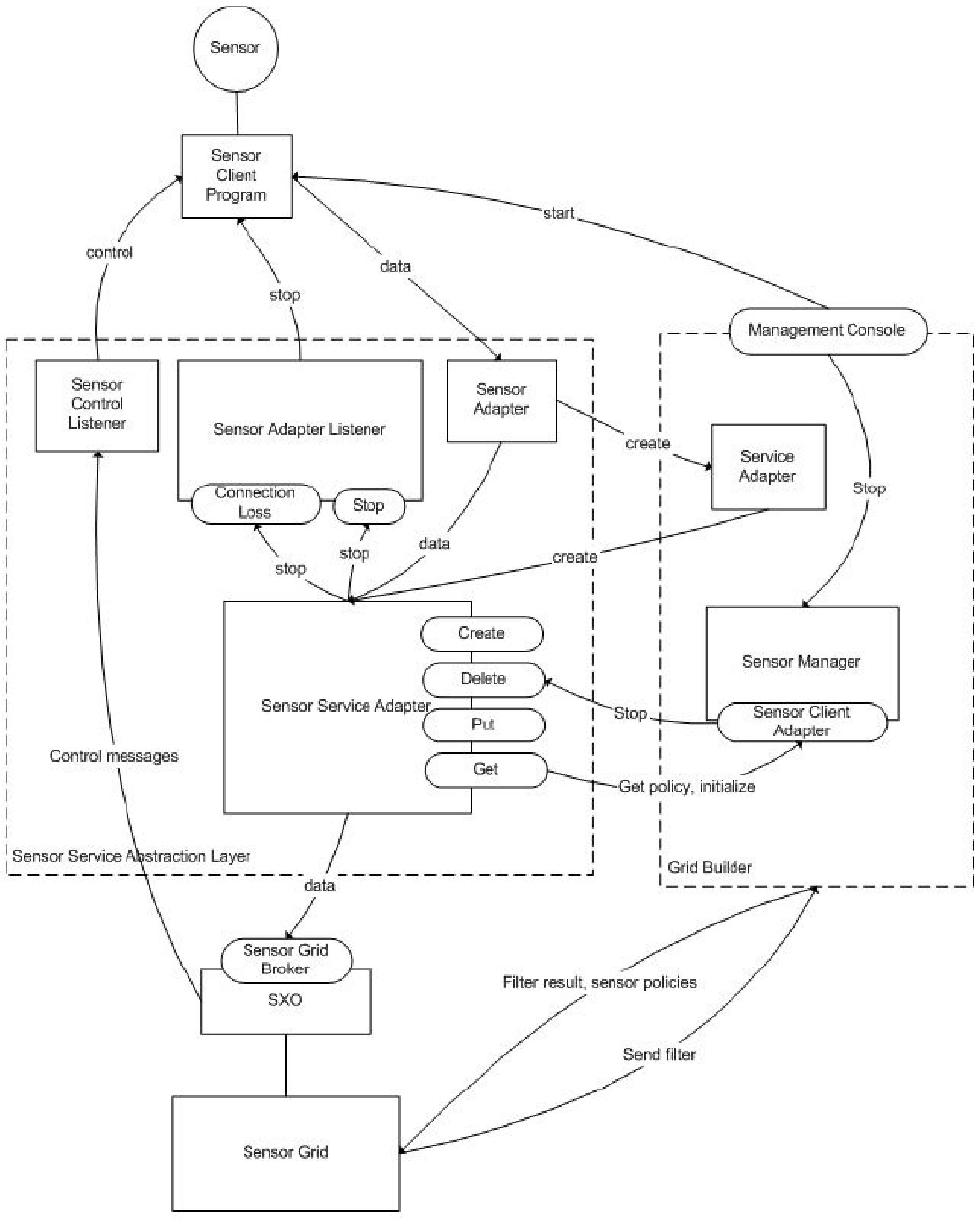


Figure 42. A detailed SSAL architecture for general sensor services

4.1.5.2.2.1 Sensor Deployment

To deploy a sensor, the corresponding SCP has to instantiate an SA which, in turn, notifies SCMW of its presence and data publishing. It also has to implement a Sensor Control Listener (for receiving control messages) and a Sensor Adapter Listener (for actions such as terminating SCP). The SCP can either be started by a manner decided by the sensor developer (e.g., run a .bat script), or it can be embedded in SCMW so that it can be started by GB's Management Console.

4.1.5.2.2.2 Data Publishing

SCP is responsible for collecting data from the sensor, and then publishing it through the SA. SA, in turn, forwards the data to the corresponding SSA, and finally to all applications that have subscribed to its data.

4.1.5.2.2.3 Performing Actions on Sensor Client Program

Sometimes the user may want to perform some actions remotely on the SCP, such as pausing or terminating the SCP. SCP listens for these actions through the Sensor Adapter Listener. Currently, there is only one action supported by SCMW – terminating the SCP.

4.1.5.2.3 SSAL Architecture for Computation as a Sensor Service

Architecturally SSAL for Computational Service combines SSAL for normal sensors and SCMW API since it needs functionalities from both sides. **Figure 43** shows SSAL for Computational Service. As shown, components of the SCMW API are integrated with components of the original SSAL and some new modules to form the SSAL for Computation as a Sensor Service. The extension of SSAL to cover computation as a sensor service significantly broadens the applicability of the Sensor- Centric Grid of Grids and eases the integration of new or legacy system of systems with sensor-centric applications.

The following subsections explain the message flow for some operations of Computational Services.

4.1.5.2.3.1 Sensor Deployment

To deploy a Computational Service, the corresponding SCP has to instantiate a Sensor Client Adapter which notifies SCMW of its presence and of various sensor related operations such as data publishing, subscribing data from source sensors and sending control messages to source sensors. It also has to implement a Sensor Control Listener (for receiving control messages) and a Sensor Adapter Listener (for actions such as terminating SCP) as what normal sensors do.

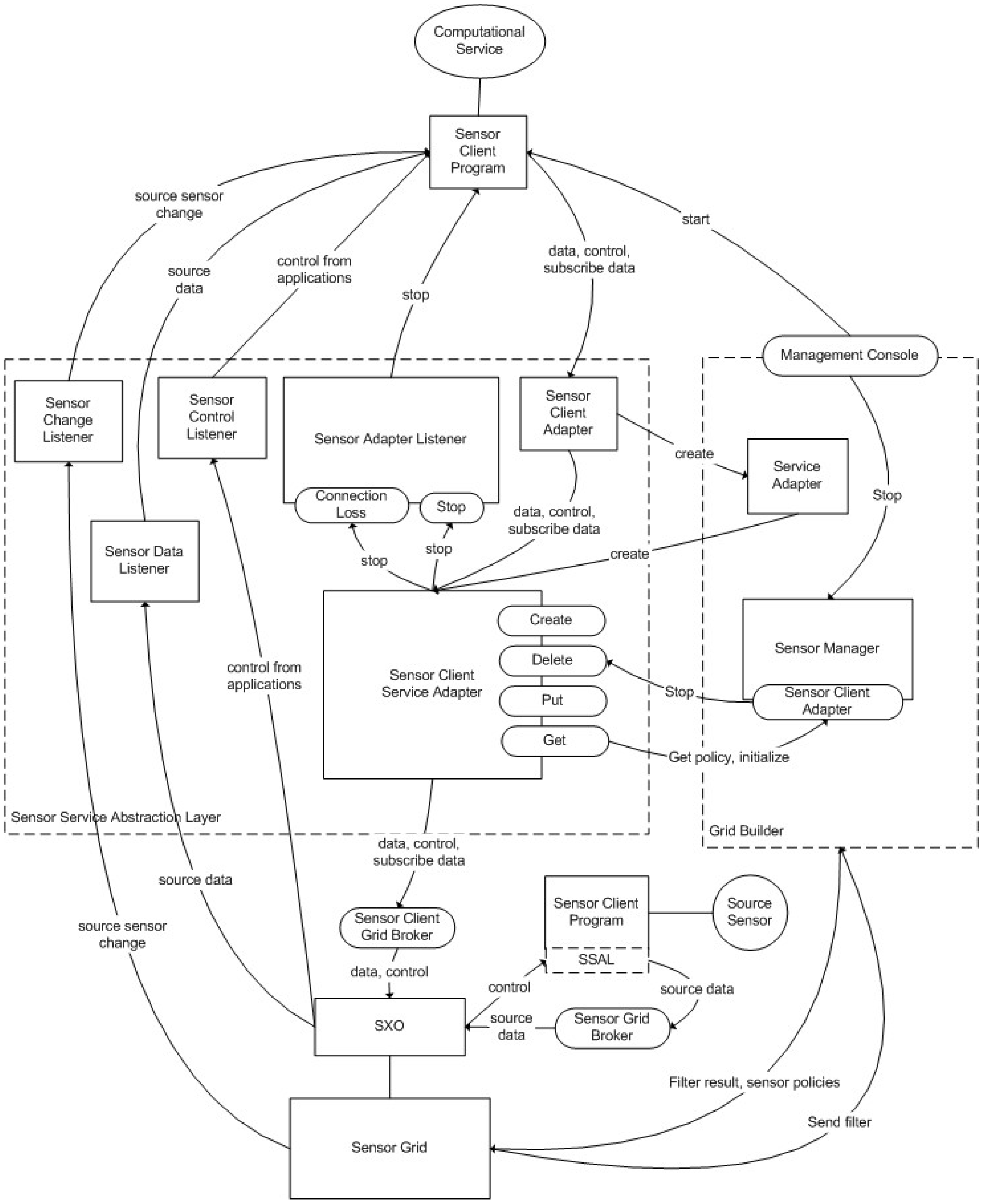


Figure 43. A detailed SSAL architecture for computation as a sensor service

4.1.5.2.3.2 Subscribe Sensor Data

Since Computational Services take input from other sensors (source sensor), they have to subscribe data from other sensors in a similar way to applications. To subscribe data, the SCP of a Computational Service has to invoke functions of Sensor Client Adapter which in turn setup the connections. SCP has to implement the Sensor Change Listener and Sensor Data Listener interfaces. Whenever the state of source sensor changes (e.g., online to offline) the SCP will be notified through Sensor Change Listener. Similarly SCP will be notified for data arrival through Sensor Data Listener.

While the previous sections (4.1.1 through 4.1.5) describe the results and accomplishments associated with the Sensor Cloud implementation, the following sections (4.1.6 through 4.1.8) describe the R&D related to the baseline SCGMMS enhancements that supported the sensor cloud implementation.

4.1.6 Core SCGMMS Enhancement

4.1.6.1 Interim Standalone Rich-Client

The initial AFRL SBIR Phase I and Phase II efforts “Grid of Grids for Information Management” and “Net-centric Sensor Grids” resulted in the design and development of a research prototype for SCGMMS with a UDOP and a Community Collaboration Grid Building Tool.

The original demonstration client software for the collaborative sensor grid application developed to prove the concept of SCGMMS was based on Anabas proprietary collaboration technology, interfaces and software. To support IU, that led a major subtask to enhance the core SCGMMS, Anabas initiated an effort to isolate the dependency of the former demonstration client on Anabas proprietary collaboration technology, interfaces and software; and developed an interim, standalone, rich-client for sensor grid demonstration that uses the SCGMMS but not the Anabas proprietary collaboration technology and interfaces. There were two objectives associated with this de-coupling effort. One was to reduce code complexity for IU by removing certain code that was incorporated for the purposes of demonstration but not needed for core SCGMMS and its further enhancement. The other objective was to maintain exactly the same look-and-feel, graphical user-interfaces and layout, and SCGMMS demonstrable features, including the initial support of UDOP, sensor filtering and hierarchical sensors, in the interim, standalone, rich-client so that there was a demonstration client available throughout the research.

4.1.6.2 Sensor Streams Support for Web Clients

To augment rich SCGMMS clients like the original standalone demonstration application which illustrates some level of support for sophisticated features like UDOP and sensor property filtering; agile, light-weight Web clients running on mobile platforms (like tablets or smartphones) was an important class of devices to be supported. For this purpose the research team developed an illustrative support for Asynchronous JavaScript and XML (AJAX)-based Web client interactions with SCGMMS sensor services as a preliminary demonstration of one way to add such capability in the context of SCGMMS and associated sensor APIs. This was done to serve as an example for IU regarding how Web servers could be integrated with SCGMMS.

AJAX is the use of the XML HTTP Request objects to communicate with server-side scripts. AJAX can send as well as receive information in a variety of formats, including JavaScript Object Notation (JSON), XML, Hypertext Mark-up Language (HTML), and even text files. AJAX's most appealing characteristic, however, is its "asynchronous" nature. It enables clients to asynchronously poll for server-side events. By polling, server events can be queued and delivered to the browser on each poll interval, which emulates server initiated communications and provides real-time message delivery within the bounds of the poll interval.

Comet, which is also known as AJAX Push or Reverse AJAX, introduces techniques that depart from the HTTP communications model by enabling a "push"-style of communications over HTTP. Comet defines several techniques that allow the server to send information to the browser without prompting from a client. With the help of an additional HTTP connection, Comet can even facilitate bi-directional communications over two HTTP connections. Comet attempts to deliver "push" communications by maintaining a persistent connection or long-lived HTTP request between the server and the browser. This connection allows the server to send events, initiated by the client to the browser. Upstream requests can be issued by the browser to the server, and made over an additional HTTP connection. Thus, Comet can facilitate bi-directional communications over two HTTP connections. However, the maintenance of these two connections introduces overhead in terms of resource consumption on the server.

For the purpose of illustrating a viable architecture for integrating Web servers with SCGMMS, researchers chose to extend a Comet-capable Web Server, using the Apache Tomcat Comet module, as a sensor application. An AJAX client that could receive GPS sensor streams was developed to verify the approach successfully. In the preliminary illustrative implementation, researchers developed sensor stream retrieval but not control data communication. The code was submitted to IU as a sample implementation.

4.1.7 Collaborative Sensor-Centric Grid Framework

In order to generate and measure collaborative sensor-centric grid application traffic on distributed clouds, researchers first needed tools to build a sensor-centric grid, and to deploy and manage sensors. Instead of developing new tools and technology for building a sensor-centric grid and deploying and managing sensors, the team reused some of those capabilities developed for an earlier project, namely a collaborative sensor-centric grid framework [9]. The framework supports the integration of a sensor-centric grid with collaboration and other grids, and provides a sensor interface and sensor-centric application interface. The framework also includes GB, a grid builder tool, for building, deploying, discovering and managing grid services and local and remote sensors.

GB follows the idea of constructing grids of grids, which assembles a multitude of subgrids into a mission-specific grid application. GB is a sensor management module which provides services for (a) defining sensor properties, (b) deploying sensors according to defined properties, (c) monitoring deployment status of sensors, (d) remote management irrespective of the locations of deployed sensors, and (e) distributed management irrespective of the location of the operator/user. Sensor streams are being shared in real-time with any sensor-centric applications that are developed using the API provided by the framework. A deployed sensor-centric grid communicates with (a) deployed sensors irrespective of sensor locations, (b) deployed sensor-centric applications irrespective of application locations, and (c) Grid Builder to mediate the collaboration among these three modules. In this framework, a primary function of a sensor-centric grid is to manage and broker message flows for sensor data and controls.

A typical scenario of a collaborative sensor-centric application using the framework encompasses a global deployment of a large number of sensors of different types. Each sensor (for examples, video, GPS, video/audio, sound, light, temperature, gyroscope, ultrasonic, or RFID) gathers data from its environment and publishes it in real-time to a sensor-centric grid via a sensor adapter architecture. Some types of sensors can subscribe to other sensors' published data in the sensor-centric grid and provide filtering services, the results of which are published to the sensor-centric grid like any other sensors. A collaborative sensor-centric application provides the application logic and user-interface to orchestrate and manage real-time collaboration among only those sensors of interest for timely decision-support.

A demonstrative illustration of a sensor-centric application over the public Internet for collaborative, real-time sensor control and video motion detection was described in [9]. The demonstrative scenario involved the deployment of sensors in California, Indiana and Hong Kong. We summarily depict the scenario as shown in **Figure 44** and **Figure 45**. Figure 44 shows some sensors and a robot with the sensor payload deployed in Hong Kong for a collaborative sensor-centric application demo, where one of the sensors is a Lego NXT Tribot. Figure 45 shows a snapshot of the Hong Kong-deployed Tribot being controlled in real-time by a California-deployed WiiMote (Wii remote control) sensor. The sensor data is superimposed with the live filtering of the video stream from the Hong Kong-deployed webcam sensor by the Indiana-deployed software-based video motion detection sensor, which draws a bounding box around the area where motion is detected.

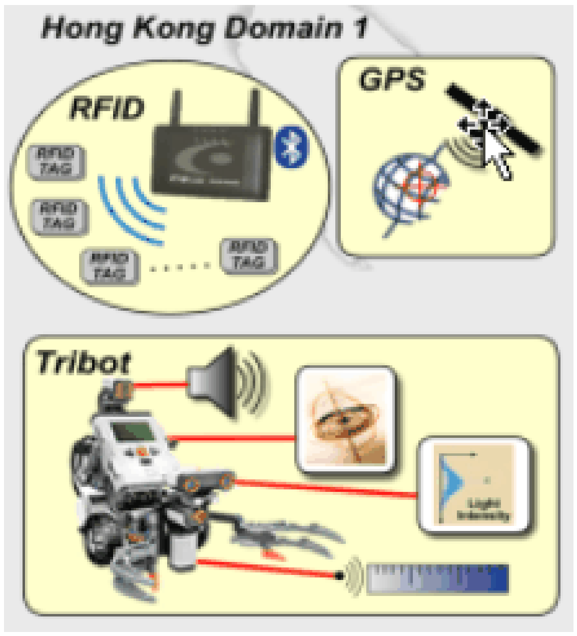


Figure 44. Collaborative Sensor-Centric Application Demo

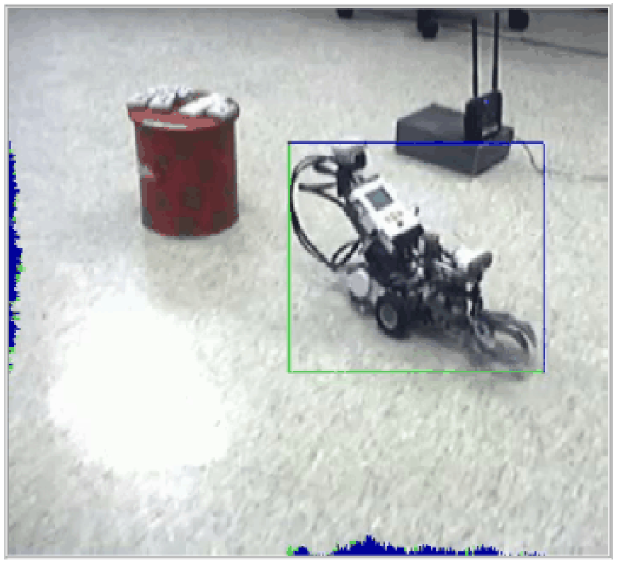


Figure 45. Real-Time Collaborative Tribot Control and Motion Sensing

To lay the foundation for implementing a sensor cloud, another aspect of this research examined heterogeneous and distributed clouds running on commercial Amazon EC2 clouds, FutureGrid cyber-infrastructure, and private clouds. The research team developed another sensor-centric application using the framework. Researchers also ported GB to FutureGrid which enabled them to build a sensor-centric grid, deploy sensors and sensor-centric applications to generate, measure and analyze specific application-level performance on FutureGrid distributed clouds.

This phase of the Sensor Grid research has been very valuable in laying the foundation to building a robust functioning sensor grid that incorporates various aspects of trustworthiness. Some practical achievements were made with working with the sensor grid middleware and key feedback to the middleware developers was used to make significant improvements in the mobility and portability of the sensor grid middleware interface for more efficient operation on mobile devices such as smartphones, mobile and stationary sensors, small portable processors (e.g., GumStix), etc. The next section describes some of the accomplishments related to porting GB to Future Grid. Some of the members of the AMSA TO4 research team were also part of the FutureGrid research team. This relationship facilitated the use of FutureGrid for experiments related to porting the Sensor Grid Middleware to FutureGrid for the sensor cloud experiment.

4.1.8 Cloud Infrastructure for Sensor Grids

4.1.8.1 FutureGrid– A National Cloud Infrastructure

FutureGrid [2] is a part of the TeraGrid [13]. The aim of FutureGrid is to support the development of new system software and applications that can be simulated in order to accelerate the adoption of new technologies in scientific computing. The project has several computing clusters at different locations with a sophisticated virtual machine and workflow-based simulation environment to support research on cloud computing, multicore computing, new algorithms and software paradigms.

Unlike production cloud systems like the Amazon EC2, Microsoft Azure or Google App Engines for commercial applications, or TeraGrid for scientific computing, FutureGrid, by contrast, is oriented towards developing tools and technologies rather than providing production computational capacity [14].

FutureGrid is an infrastructure comprising currently approximately 4,000 cores at six sites - Indiana University (11 Teraflop IBM 1024 cores, 7 Teraflop Cray 684 cores, 5 Teraflop Disk Rich 512 cores), University of Chicago (7 Teraflop IBM 672 cores), University of California San Diego Supercomputing Center (7 Teraflop IBM 672 cores), University of Florida (3 Teraflop IBM 256 cores), Purdue University (4 Teraflop Dell 384 cores) and Texas Advanced Computing

Center (8 Teraflop Dell 768 cores) - connected by a high-speed, network which is dedicated except for a public link the to Texas Advanced Computing Center. It is an experimental testbed that could support large-scale research on distributed and parallel systems, algorithms, middleware and applications. **Figure 46** shows the connectivity of the six sites.

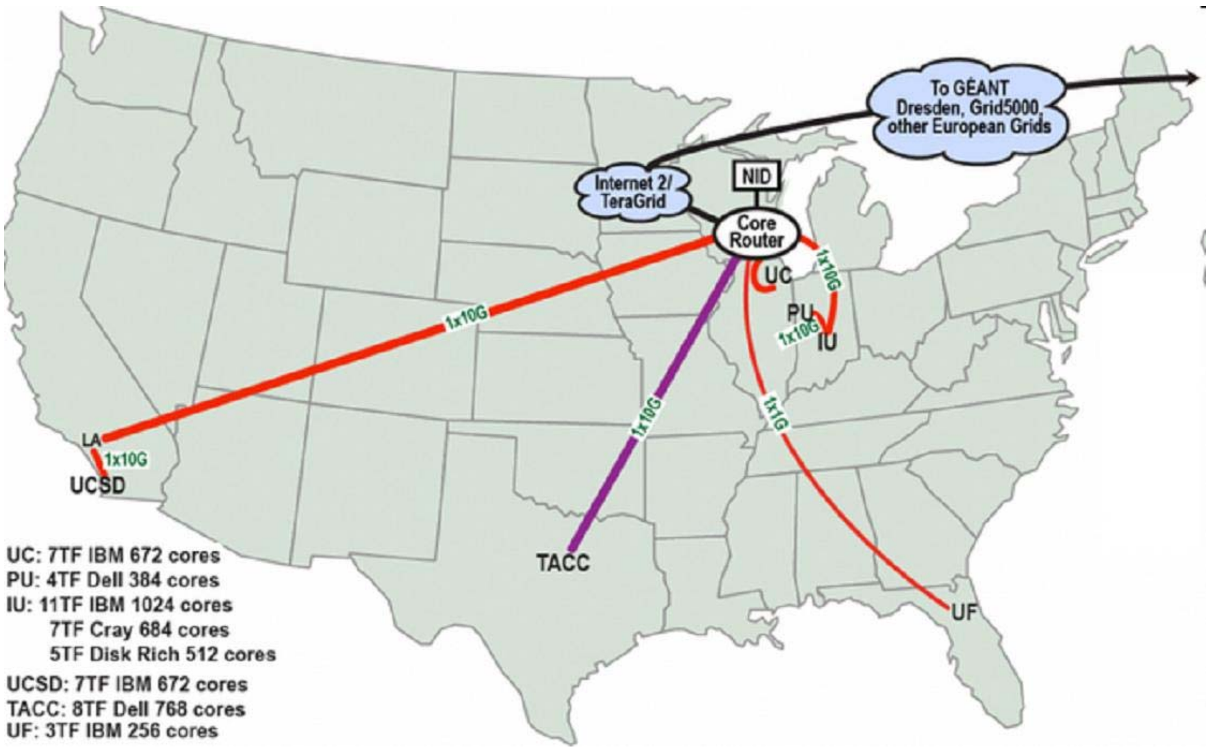


Figure 46. FutureGrid Connectivity and Capacity (courtesy of FutureGrid)

FutureGrid includes services accessible to users to run High Performance Computing (HPC) jobs such as MPI or OpenMP. It also supports several Grid and Cloud environments including the Eucalyptus and Nimbus Clouds.

Eucalyptus [15, 16] is an open source software platform that implements an Infrastructure-as-a-Service (IaaS)-style cloud computing. Eucalyptus provides an Amazon Web Services (AWS)-compliant, EC2-based web service interface for interacting with the cloud service. Additionally, Eucalyptus provides Walrus, an AWS storage-compliant service, and a user interface for managing users and images.

Nimbus is an open source toolkit that allows one to turn a cluster into an IaaS cloud [17]. Nimbus on FutureGrid allows users to run virtual machines on FutureGrid hardware. A Nimbus account user can easily upload custom-built virtual machine (VM) image or customize an image provided by FutureGrid. When a VM is booted, it is assigned a public Internet Protocol (IP) address (and/or an optional private address). The VM is accessible by logging in as root via

Secure Shell (SSH). Users can then run a service, perform computations, or configure the system as desired. After using and configuring the VM, the modified VM image can be saved to the Nimbus image repository.

4.1.8.2 Amazon EC2 Cloud– A Global Commercial Cloud Infrastructure

Amazon EC2 is a Web service that delivers resizable, scalable, pay-as-you-go compute capacity in the cloud. It is a central part of the commercial AWS cloud computing platform. The Amazon EC2 uses Xen for its underlying virtualization technology. In August 2008, the Amazon EC2 was released as full product for public release. It is the first commercial cloud infrastructure provider in the industry.

Amazon EC2 uses data centers in several countries, thus, providing some level of location optimization and fault-tolerance possibilities to its users. Except in the U.S. where Amazon EC2 has 3 data centers – US West (North California), US West (Oregon), US East (Virginia), there is also one data center each in Asia Pacific (Tokyo, Japan), Asia Pacific (Singapore), South America (Sao Paulo, Brazil) and Europe East (Dublin, Ireland). Anabas completed cloud characterization experiments for this research effort in US West (North California), EU East (Ireland), Asia Pacific (Japan), Asia Pacific (Singapore) and South America (Brazil). The Amazon EC2 is an evolving infrastructure, yet it is the most mature commercial offering that has a Web-scale global coverage.

Experiments on characterizing performance, reliability and scalability in a globally distributed configuration help to improve understanding of potential issues on how globally deployed sensor grids may perform in real-world settings.

4.1.8.3 Private clouds– Single Organization-owned Cloud Infrastructure

There are roles for both a private cloud, which is managed and owned by an organization for its own use, and a public cloud, which is a shared infrastructure operated by other enterprises or providers. For many organizations a major advantage of using private cloud over public cloud is the ability to maintain the control and implementation of security and compliance solutions that augment current cloud capabilities of other cloud solution providers. Private clouds could be deployed as on-premise to gain maximum control of security, or in data centers using dedicated resources.

There is no existing single open standard or a de-facto standard for building a private cloud computing platform. The Amazon AWS API is a market leader in the real-world production-level deployment. Compatibility to the Amazon AWS API is an important consideration for portability. OpenStack, developed and open sourced by NASA and Rackspace, has been adopted by IBM and HP for enterprise business cloud initiatives. OpenStack is experiencing increased visibility in the cloud developers and users communities. While VMware is an industry leader in virtualization software, HP's private cloud solution favors the use of Kernel-based Virtual Machine hypervisor for virtualization. Citrix is pushing CloudStack within the Apache Software Foundation to establish Xen as the virtualization technology of choice on top of its support of OpenStack as an IaaS technology. On the other hand, Rackspace's private cloud solution uses OpenStack as the cloud operating system and VMware for virtualization. Eucalyptus is another IaaS software that support building public and private clouds. Eucalyptus (standing for Elastic Utility Computing Architecture Linking Your Programs To Useful Systems) was once the default cloud module for Ubuntu Linux. Since mid-2011 OpenStack has replaced Eucalyptus as the default cloud module for Ubuntu.

4.1.8.4 Hybrid Cloud on Heterogeneous Cloud Technologies

Each organization has different combination and level of operation requirements such as security, scalability, and QoS for its overall IT needs. Where the use of cloud solutions is appropriate these requirements could be best served by leveraging private cloud, community cloud, public cloud or some combinations of them. Private cloud is an infrastructure solely operated by or for a single organization. Community cloud is a shared infrastructure among several organizations, coming from specific community of interest and with common goals or concerns. Public cloud is generally the Web-scale commercial cloud infrastructure operated for and use by the public under commercial terms.

A hybrid cloud is a composition of multiple clouds that remain unique entities but are integrated together at some levels to meet particular operation requirements of organizations. The concept of a hybrid cloud is starting to establish itself as the de-facto enterprise cloud computing model in which some form of private clouds, community clouds, and public clouds will be leveraged together with traditional IT resources.

The team's research focus was on measuring the characteristics of hybrid clouds that uses a plethora of heterogeneous cloud technologies for SCGMMS-type of applications.

4.1.8.5 Cloud Experiments

Without loss of generality for the objectives of the research the team chose to perform the cloud experiments on the national-scale FutureGrid as a community cloud, Web-scale Amazon EC2 as a public cloud, and a small organization-scale OpenStack-based infrastructure as a private cloud.

Experiments were performed on each of the cloud choices as well as some configuration of inter-clouds to form scenarios of a hybrid cloud.

To ensure acceptable precision of timing measurements in a distributed environment, the team used the network time protocol (NTP) date command to synchronize the cloud instances launched in the experiments with a time server in Chicago. In a Linux environment, which is the case for these experiments, the use of the NTP algorithm can usually maintain time synchronization to within 10 milliseconds over the public Internet.

For network-level measurement, the team used the ping [18] and iperf [19] commands, both are commonly used by network administrators to monitor network characteristics. Ping is used to test the reachability of a host on an IP network and measure round-trip transmission time for Internet Control Message Protocol (ICMP) echo request packets to and an ICMP response from the target host. In the process ping records any packet loss. Iperf is used to create TCP and UDP data streams, and measure network throughput.

For transport-level measurement, the team used NB messages modeled after typical multipoint video conferencing traffic. NB servers work as an overlay transport layer to applications by taking care of all the communication among nodes composing the application using it NB is a middleware working as a glue connecting remote parts of a distributed application.

For application-level traffic generation and data gathering, the team used the collaborative sensor-centric grid framework and the grid builder tool. To investigate scalability issues, it was not practical to deploy real sensors in a large scale. Instead, the team deployed virtual sensors. The collaborative sensor-centric grid framework supports development and deployment of real and/or virtual sensors. As an initial study on a multi-point distributed cloud, the team deployed virtual GPS sensors only, even though they had developed virtual sensors for RFID and WiiMote.

4.1.8.5.1 FutureGrid As A Representative Community Cloud

In the study, the team used up to four clouds on FutureGrid. The clouds that were used were the Hotel (in University of Chicago running Nimbus), Foxtrot (in University of Florida running Nimbus), India (in Indiana University running Eucalyptus) and Sierra (in San Diego Supercomputing Center running Eucalyptus). The distributed clouds scenario setup either involves pairs of clouds or a group of four clouds. The team chose m1.xlarge instances in the Eucalyptus cloud (each m1.xlarge instance is approximately equivalent to a 2-core Intel Xeon X5570 with 12 GB RAM) and 2 cores with 12 GB RAM in Nimbus. The selection of m1.xlarge

VM in Eucalyptus is to ensure the Eucalyptus VMs were used for heterogeneous distributed cloud experiments were about the same level of computing resource as those in Nimbus.

4.1.8.5.1.1 FutureGrid Network-Level Measurement

The team ran two types of experiments. They were (a) single-pair of cloud instances, one instance on each cloud, using iperf for measuring bi-directional throughput between all 2-combination distributed clouds of the set of four clouds selected (Hotel, Foxtrot, India, and Sierra); and (b) single-pair of cloud instances, one instance on each cloud, using the ping command together with the iperf command for measuring packet loss and round-trip latency under loaded and unloaded network between all 2-combination of the set of four clouds selected.

Figure 47 shows measured total bi-directional throughput using a range of one to sixty-four iperf connections for all 2-combination distributed clouds of the set of four selected clouds. The legend of Figure 47 shows all six combinations of 2-combination distributed clouds in the setup.

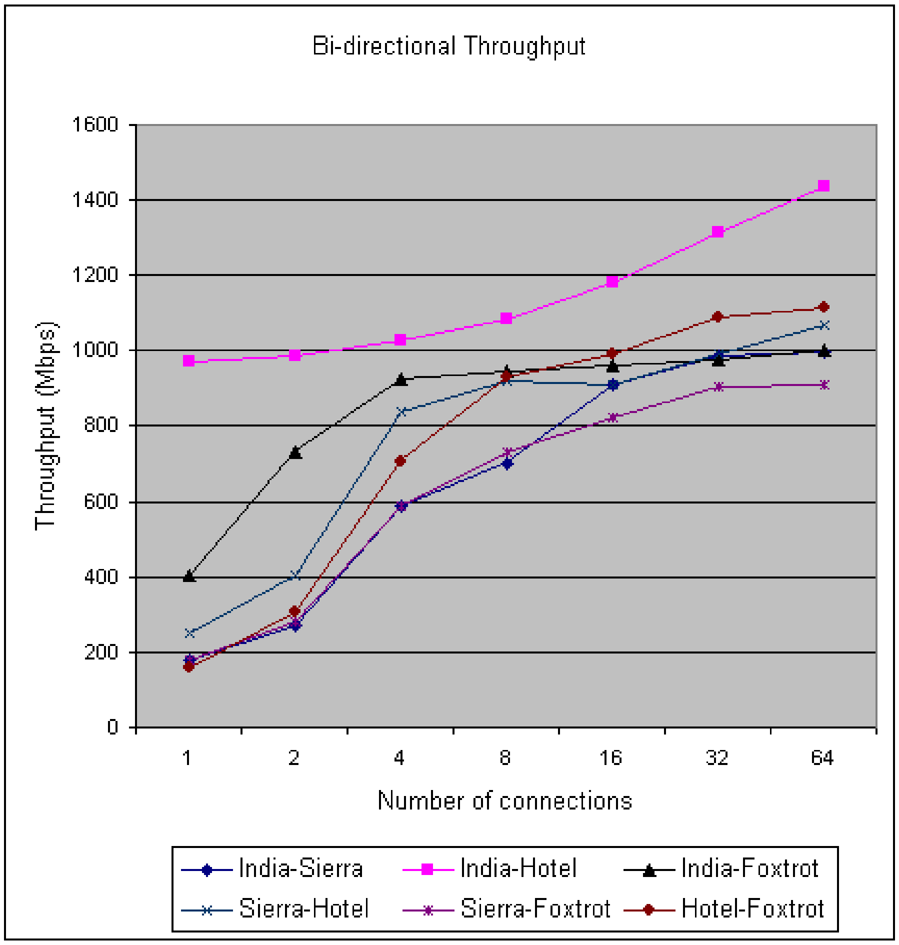


Figure 47. Throughput Between Distributed Clouds.

While the maximum bi-directional throughput between any 2-combination ranges from 900 Mbps (on Sierra/Foxtrot pair) to 1,400 Mbps (on India/Hotel pair), the team found the total iperf throughput in FutureGrid was over 800 Mbps when they connected any pair of cloud instances on distinct clouds with more than 16 connections in each direction.

The team used the ping tool to measure network latency and packet loss between two clouds. Figure 47 shows the throughput between any two clouds in the experiments either levels off or start to level off at 32 iperf connections for all but the connection between India and Hotel.

For comprehensiveness, the number of iperf connections should be increased up to the point the network is saturated to explore the elasticity of the current state of the FutureGrid network. The team used iperf with 32 connections only to generate relatively heavy traffic of a loaded network for their initial study. They reported measured network latency and packet loss in the connections between all 2-combination distributed clouds for both loaded and unloaded networks.

The results (see **Table 3**) show ping packet loss rates in unloaded network for all the 2-combination of clouds were 0%; while the highest ping packet loss rate was 0.67% between the India/Hotel pair. The results indicate a highly reliable FutureGrid network under the experimental conditions.

Table 3. Inter-cloud Ping Packet Loss Rate

Instance Pair	Unloaded Packet Loss Rate	Loaded Packet Loss Rate
India-Sierra	0%	0.33%
India-Hotel	0%	0.67%
India-Foxtrot	0%	0%
Sierra-Hotel	0%	0.33%
Sierra-Foxtrot	0%	0%
Hotel-Foxtrot	0%	0.33%

For baseline information, the team measured ping round-trip latency between 2 cloud instances on Sierra for the unloaded case and loaded cases with 16 and 32 connections before conducting the same experiment on distributed clouds. They found latencies for the unloaded and the two loaded cases between two virtual machines communicating on the same cloud no higher than 1.18 milliseconds. Thus, they reasonably assumed for the ping experiments on distributed clouds the measured round-trip latencies were mainly due to the distance between clouds. Virtual machine overhead was negligible in these experiments.

Ping round-trip latency for all six combinations of pairs of clouds was measured. The team found the lowest average round-trip latency of about 18 milliseconds between India and Hotel in a loaded condition (see **Figure 48**). India and Hotel had the shortest distance between any two of the four clouds; and thus, was expected to show the lowest round-trip latency.

The team observed the highest ping round-trip latency in a loaded network condition was about 145 milliseconds on the Sierra and Foxtrot connection (see **Figure 49**). Although the inter-cloud latency between Sierra and Foxtrot was the highest due to its longest distance between any two of the four selected clouds, the team noted that a round-trip latency below 300 milliseconds still met a requirement for acceptable quality of service for collaboration applications with stringent network requirement like that of VoIP [19].

Overall, the limited initial results indicated that FutureGrid can sustain at least near 1 Gbps inter-cloud throughput and is a reliable network with low packet loss rate.

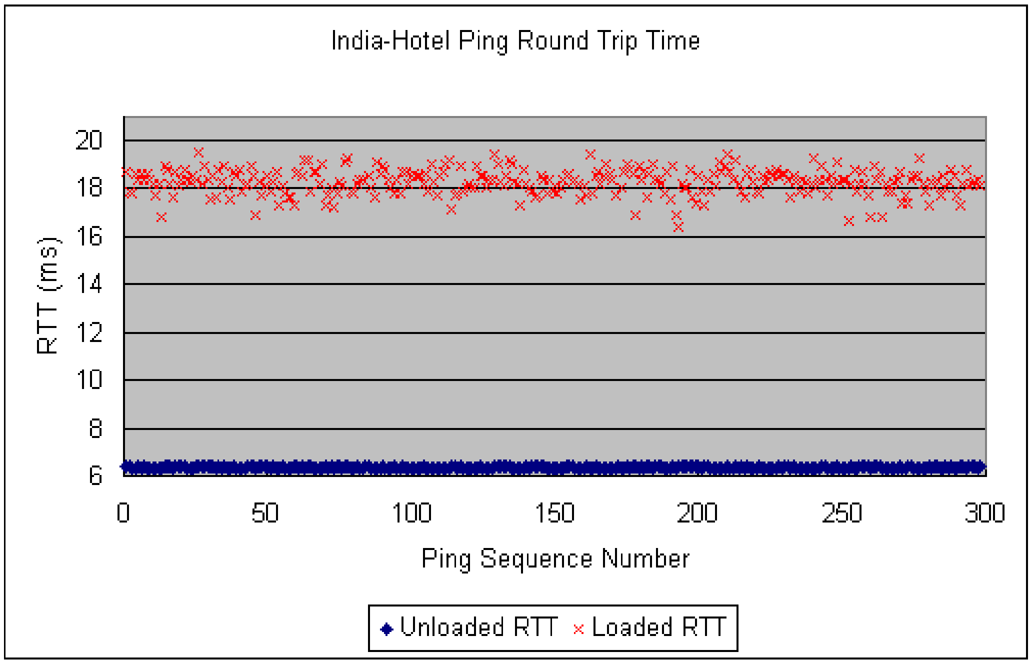


Figure 48. Ping Round-trip Latency between India and Hotel

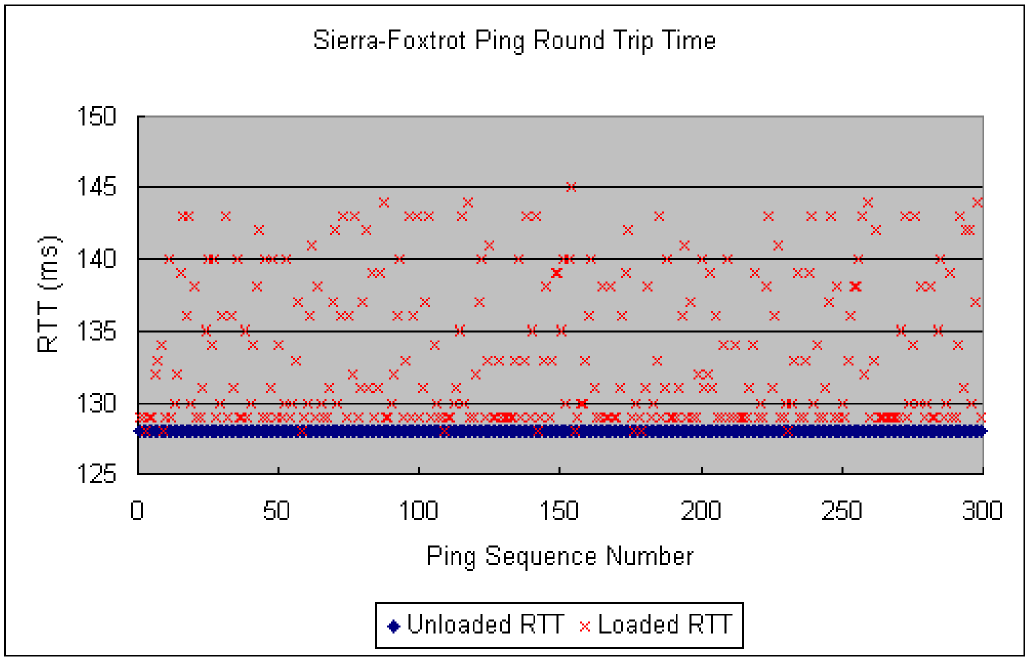


Figure 49. Ping Round-Trip Latency Between Sierra and Foxtrot

4.1.8.5.1.2 FutureGrid Message-Level Measurement

In one set of experiments, extensive measurements were taken to evaluate the performance, stability and reliability characteristics for an increasingly larger collaboration session by injecting NB messages. The team selected the Foxtrot and Hotel, both running Nimbus environment, for the 2-cloud distributed experiments. An NB instance ran on Foxtrot. Simulated multiple meetings with groups of 20 participants were run on Hotel.

Even though there was an actual multipoint video conferencing application in the Impromptu conferencing suite that could have been used to generate real video traffic, it is easier and more practical to scale the number of users/participants at the message-level using NB clients than at the application level using real cameras and people for modeling a large-scale video session.

Figure 50 shows latency data on the inter-cloud connection between Foxtrot and Hotel. The average latency incurred in a single meeting with up to about 2,400 participants was below 50 milliseconds. Average latency jumped rapidly when the number of participants in a single meeting was more than 2,400. However, if a large meeting was divided into multiple smaller ones, the team found that distributed clouds could sustain a higher aggregate total number of participants. In these experiments, the team found the average latency could be maintained below 50 milliseconds with 150 meetings, each of which had 20 participants; that is, a total of 3,000 participants.

The average latency result indicated that multiple smaller meetings balance the work of a NB broker better. Also reflected from the experiments was that there is message backlog on a single broker when there were more than 2,400 participants in a single meeting or 3,000 participants in multiple meetings. When there is message backlog on a message broker, latency will increase rapidly. Of course NB can support multiple distributed brokers to control a collaboration or sensor network, so limits shown in Figure 50 represent limits of a single broker and not of the system. Clouds are attractive as they support the auto-scaling needed to add brokers on demand.

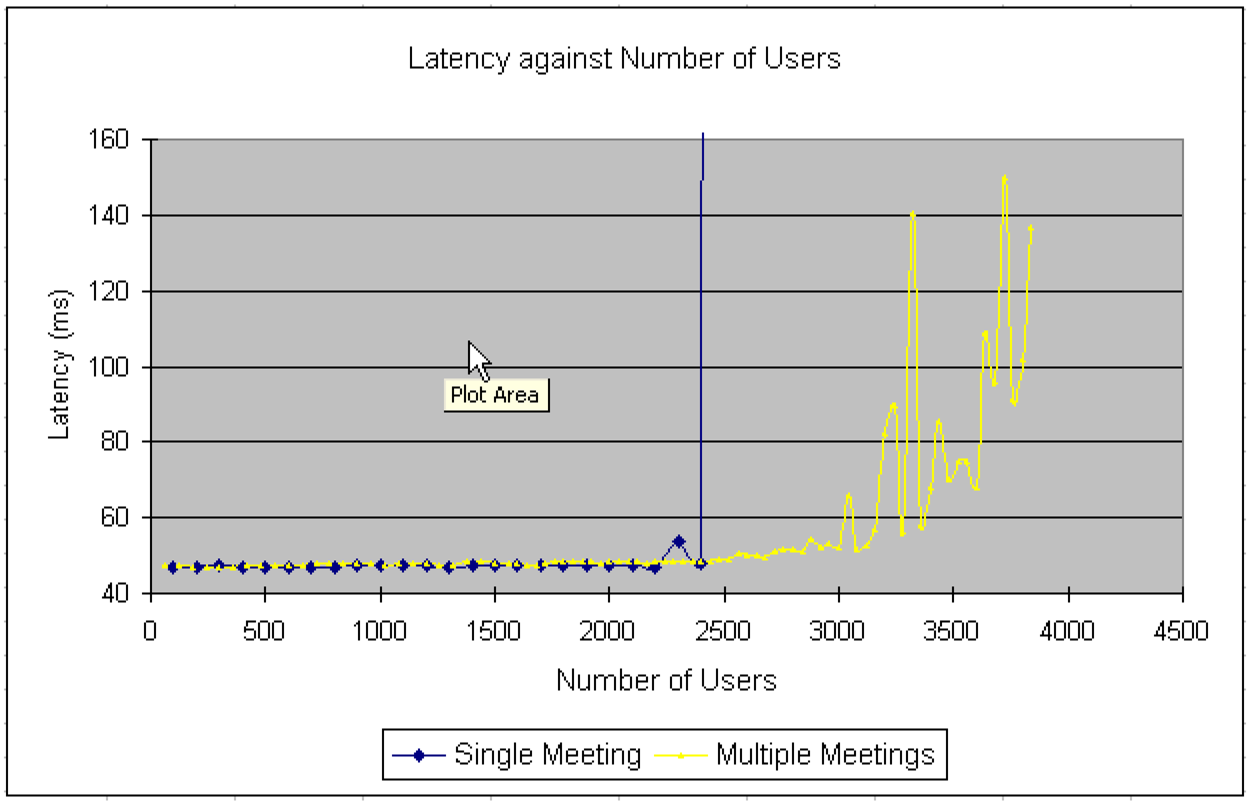


Figure 50. Average Latencies of Single and Multiple Video Meetings

Overall, these limited initial results of message-based experiments indicated that FutureGrid can sustain a throughput close to its implemented capacity of 1 Gbps between Foxtrot and Hotel. The multiple meetings experiment also showed that clouds can support publish-subscribe brokers effectively. Note the limit of around 3000 clients in Figure 50 was reported as 800 in earlier work [5]; this showed that any degradation in server performance from using clouds is more than compensated by improved server performance.

4.1.8.5.1.3 SCGMMS Application-level Measurement

In this section, the report discusses measurements of the scalability of multipoint distributed clouds on FutureGrid for collaborative sensor-centric applications. While a main objective of the research plan was to quantify the central processing unit (CPU), memory and communication requirements of a broad class of naturally distributed and highly scalable collaborative sensor-centric grid applications on the underlying distributed cloud architectures, the our initial observations of one such application, namely the collaborative sensor-centric grid framework [3], running on several distributed cloud scenarios on FutureGrid infrastructure is discussed here.

The team developed and used virtual GPS sensors that they modeled after real GPS sensors. These are functional virtual sensors with reasonable design but their implementations were not optimized in any way. Each virtual GPS sensor streams information to the sensor-centric grid at a rate of 1 message per second. A sensor-centric grid application consumed all the sensor streams and computed message latency and jitter for a range of deployed sensors.

The research team first established a performance baseline by deploying as many virtual GPS sensors as possible in one cloud instance without hitting any critical bottlenecks in CPU or RAM. When they deployed 100 virtual GPS sensors in an instance in the India cloud, the team observed the sensors continued running even though both idle CPU and unused RAM were at a critically low level, with idle CPU at 7% and unused RAM at 1 GB. Since the primary focus was on distributed cloud communication characteristics for scalable collaborative real-time sensor-centric applications, the team wanted to avoid running into CPU or RAM bottlenecks in the scalability experiment. When the number of deployed sensors in a single cloud instance was lowered to 60, the team observed idle CPU at about the 35% level.

The team conducted 2 different experiments. They were (a) establishing a baseline measurement within a single instance in one cloud only by deploying as many virtual sensors as possible; and (b) measurements of the communication characteristics by deploying up to 50 virtual GPS sensors in a single instance in each of the four selected clouds; that is, a total of up to 200 virtual GPS sensors were deployed in the experiment.

There were three important observations related to scalability that could be made. Firstly, as shown in **Figure 51**, in the case of using a single instance in one cloud only for deploying sensors, the maximum number of virtual GPS sensors that could be stretched in a deployment was 100, but the instance shows a critically high CPU and RAM utilization. Such low levels of unused resources in an instance had a high risk of running out of resources and becoming unstable. In the case of running a single instance in each of the four selected distributed clouds, the team observes a much lower level of resource utility. This provided a more stable and suitable environment for long running simulations.

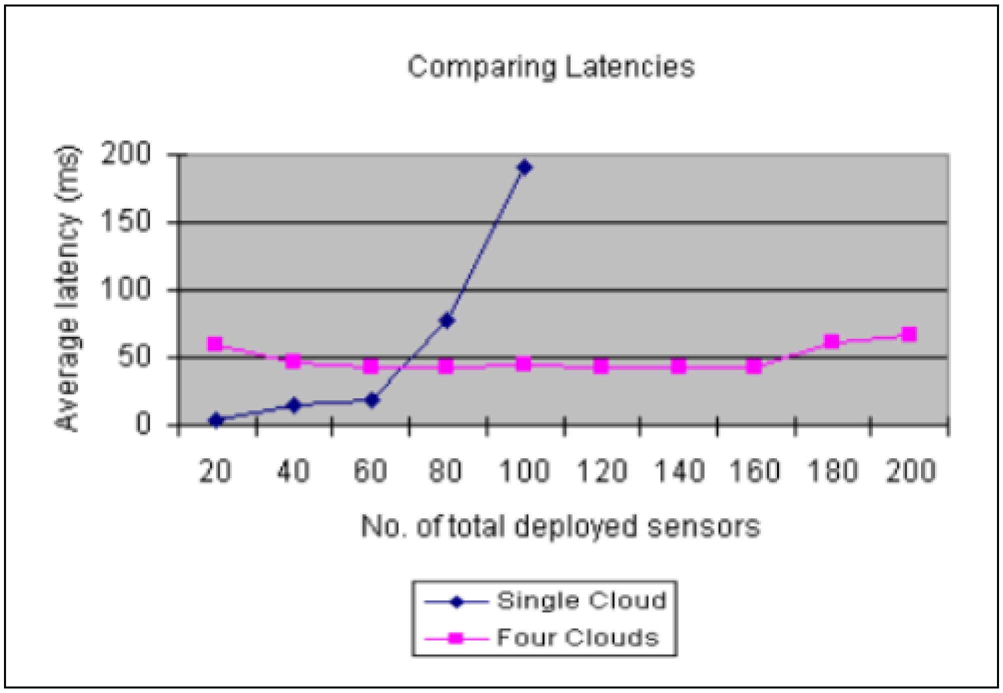


Figure 51. Comparing Average Latency of a Single Cloud and 4-point Distributed Cloud

Secondly, even though the case of using a single instance on a single cloud could have been pushed to deploy 100 virtual GPS sensors, the average latency started to grow rapidly after deploying 60 sensors. At the level of 80 deployed sensors, the average latency was higher than that of the case of the 4-point distributed cloud at the level of 200 deployed sensors. The team noticed that in the distributed case, the average latency was relatively constant and sufficiently low, even for demanding network applications like VoIP [18], and with small variations only when sensor deployment was scaled up from 20 to 200. Thirdly, a similar pattern was observed in the comparison of the average jitter for the two cases (see **Figure 52**). In the case of sensor deployment in a single instance in one cloud only, average jitter was low until after deploying 60 sensors. At the level of 80 deployed sensors, the average jitter was already higher than that of the distributed case for 200 deployed sensors.

Overall, the limited initial results indicated that distributed clouds have an encouraging potential to support scalable collaborative sensor-centric applications that have stringent throughput, latency, jitter and reliability requirements.

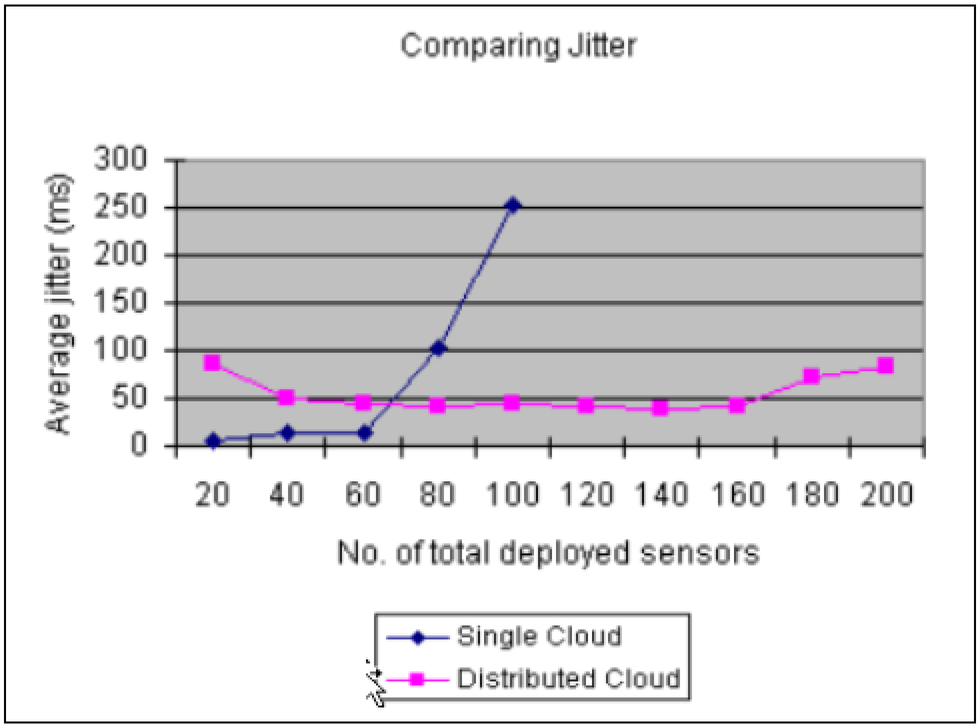


Figure 52. Comparing Average Jitter of a Single Cloud and 4-point Distributed Cloud

4.1.8.5.2 Amazon EC2 as a Representative Public Cloud

The Amazon EC2 cloud includes several regions and zones in different countries and continents. Experiments were performed on acombination of regions and zones to represent international and inter-continental scales.

4.1.8.5.2.1 Amazon EC2 Network-Level Measurement

For baseline information, the team measured ping round-trip latency, packet loss and jitter between clouds in different regions. They selected EC2 clouds in US-West (North California), Asia Pacific (Tokyo, Japan), Asia Pacific (Singapore), South America (Sao Paulo, Brazil) and EU East (Dublin, Ireland). **Figure 53** is an illustration of the pair-wise cloud connectivity map that was used for the runs. Five pairs of EC2 cloud regions were selected to give a Web-scale perspective forthe experiments. The five pairs were North California (U.S.A.) and Tokyo (Japan); Tokyo (Japan) and Singapore; Singapore and Sao Paulo (Brazil); Sao Paulo (Brazil) and Dublin (Ireland); and Dublin (Ireland) and North California (U.S.A).

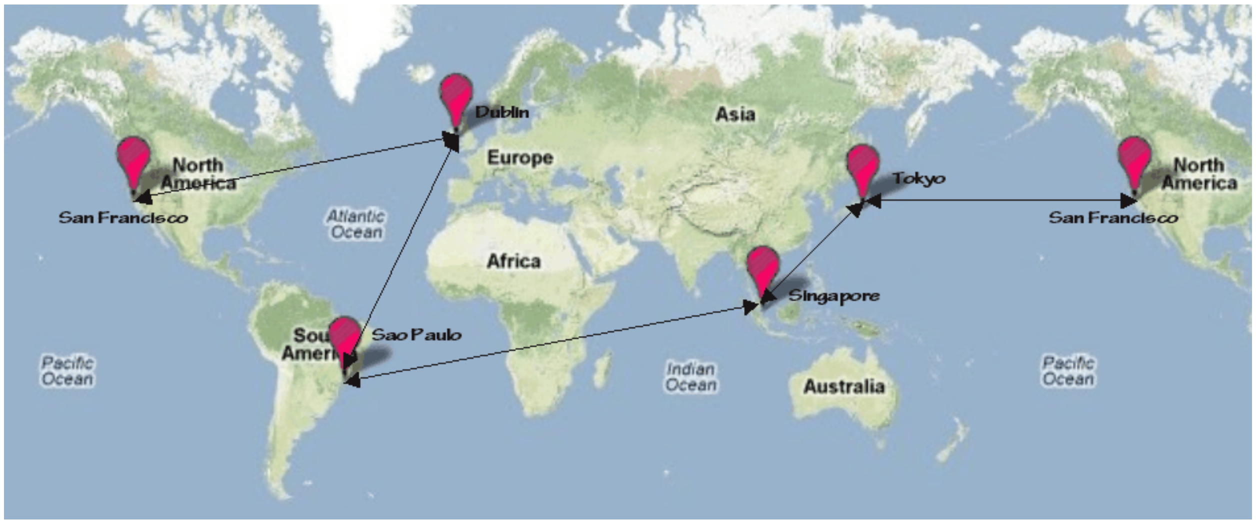


Figure 53. Web-Scale Inter-Cloud Connectivity

The Web-scale experiment results are summarized in **Table 4**.

Table 4. Web-scale Inter-cloud Latency

Instance Pair	Average RTT (ms)	Jitter (ms)	Unloaded Packet Loss Rate	Distance (miles)
N. California (USA) - Tokyo (Japan)	127.99	5.03	0	5144
Tokyo (Japan) - Singapore	80.21	3.67	0	3306
Singapore - Sao Paulo (Brazil)	362.51	6.54	0	9945
Sao Paulo (Brazil) - Dublin (Ireland)	223.21	4.88	0	5842
Dublin (Ireland) - N. California (USA)	157.17	4.63	0	5086

Ping round-trip time (RTT) and packet loss rate for all five links between cloud regions were measured, and average RTT and jitter calculated. The lowest average RTT of 80.21 milliseconds between Tokyo and Singapore was observed, corresponding to the shortest region to region distance of 3,306 miles for the links. The highest average RTT of 362.51 milliseconds was found between Singapore and Sao Paulo, corresponding to the longest region to region distance of 9,945 miles. These observations are consistent with those observed within national-scale FutureGrid. Further, as shown in the scatter plot of round-trip latency between pair-wise clouds within FutureGrid and Amazon EC2, respectively, in **Figure 54**, preliminary data indicated round-trip latency has a relatively linear relationship with physical distance between clouds. It was noted that in general the pair-wise clouds within Web-scale Amazon EC2 showed higher round-trip latency than that of the national-scale FutureGrid due to longer physical distance between clouds.

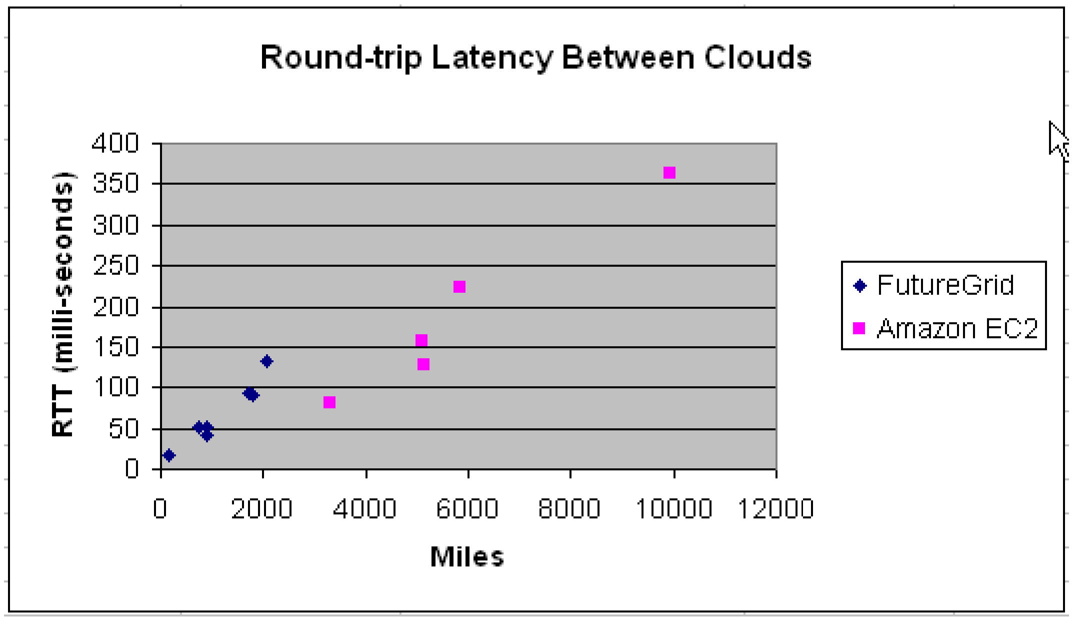


Figure 54. FutureGrid and Amazon EC2 Round-Trip Latency

For the purpose of comparison between inter-cloud throughput within FutureGrid and Amazon EC2, respectively, the earlier results obtained on EC2 throughput [10] are summarized here. Amazon EC2 has three cloud regions in the U.S. and one in Ireland. As shown in **Figure 55**. The red link connects the two regions, US-East (Virginia, U.S.) and EU-East (Dublin, Ireland), selected for the experiments.

Iperf, a commonly used network performance tool for creating TCP and UDP data streams and measuring network throughput was used with the team's EC2-US East (Virginia) and EC2-EU West (Dublin, Ireland) images. Bi-directional throughput data across the cloud trans-Atlantic link was measured and the aggregate throughput for the instance recorded.

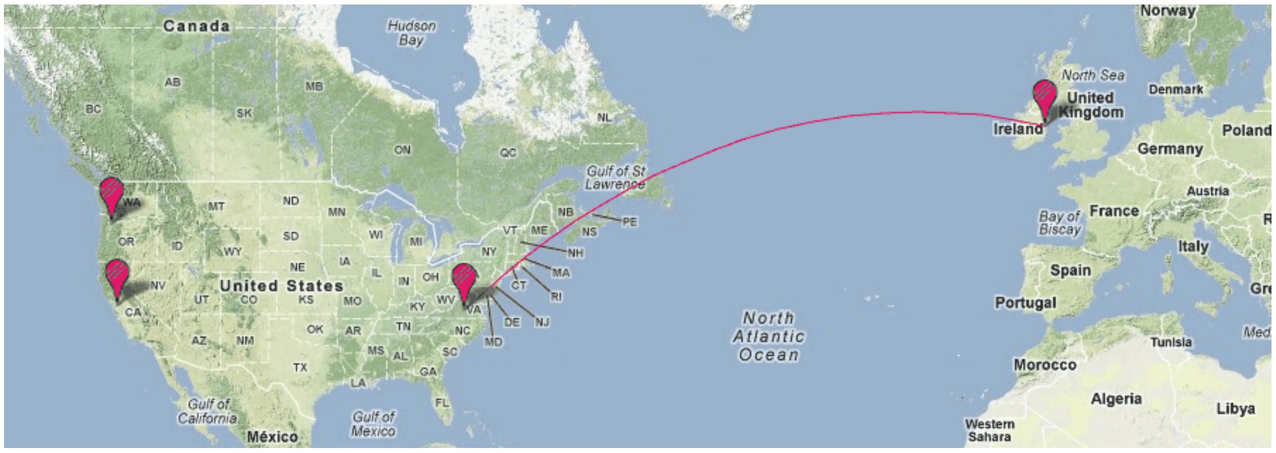


Figure 55. Amazon EC2 Regions in the U.S. and Europe

In the case when only one instance of Iperf pairs was launched in the EC2-US and EC2-EU clouds, the sub-cases of 1, 2, 4, 8, 16, 32, 64 and 128 connections were measured, accordingly. **Figure 56** shows how the inter-cloud, trans-Atlantic throughput in mbps (megabits per second) scales with the number of Iperf connections.

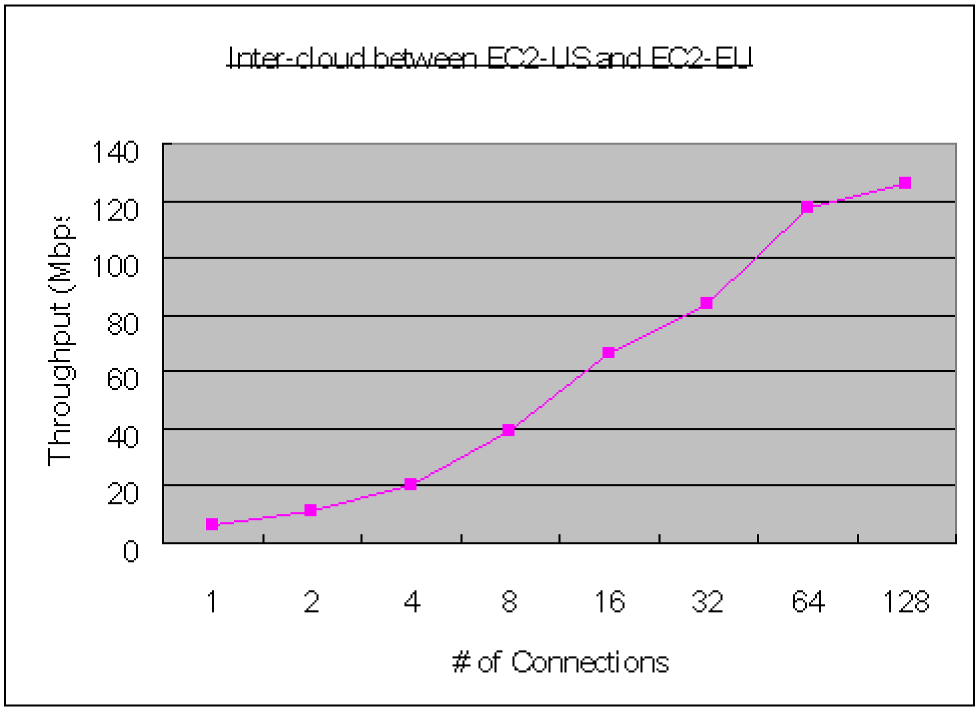


Figure 56. Throughput between 2 Trans-Atlantic Clouds

In another scenario, the number of Iperf connections was fixed at 64 while the number of instance pairs of Iperf was scaled up. **Figure 57** shows the scalability graph for up to four instances. The virtually linear scalability confirms the isolation of network resources for each instance in the cloud. Even at four instances only, the inter-cloud, trans-Atlantic throughput was already measured at nearly 500 mbps. This very large network capacity could enable large-scaled, collaboration sensor grid applications between the U.S. and Europe.

As a comparison of throughput measurements shown in Figure 47 for FutureGrid and the US-East and EU-West regions in the Amazon EC2, the maximum bi-directional throughput between any 2-combination of FutureGrid clouds ranged from 900 mbps (Sierra/Foxtrot) to 1,400 mbps (India/Hotel). Amazon’s US-East and EU-West inter-cloud sustained a throughput of 126 mbps at 128 Iperf connection. However, the team noted that the maximum sustainable throughput had not been reached in the EC2 experiments reported in [10].

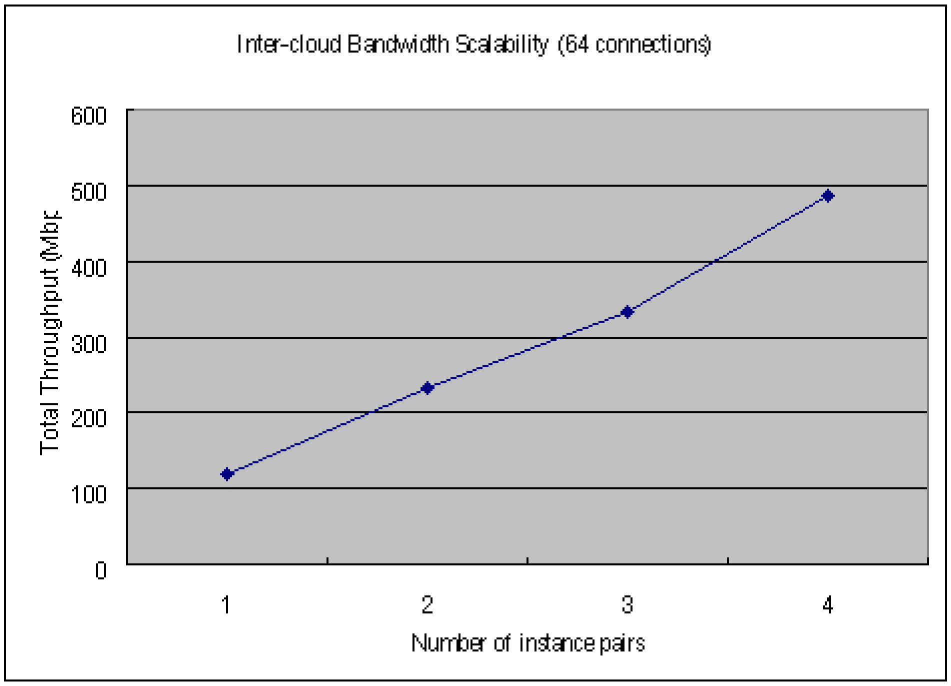


Figure 57. Scalability of Total Inter-Cloud Throughput

4.1.8.5.2.2 Amazon EC2 Message-Level Measurement

The research team compared message-level measurements taken on FutureGrid as shown in Figure 50 to some similar earlier experiments taken on Amazon EC2 as reported in [10]. In the Amazon EC2 experiments, the team also mimicked some characteristics of VoIP conferencing traffic at the message level in a conservative manner. The experiments were performed on EC2 US-East (Virginia) and EU-West (Dublin, Ireland).

An NB server was launched on EC2 EU-West while VoIP conferencing participants who were message publishers and subscribers were launched on EC2 US-East. Since all VoIP participants were in US-East, each modeled VoIP message had to traverse twice the distance between US-East and EU-West in order before arriving at a participant. Round-trip message latencies were captured. Minimum, maximum, and average round-trip message latency as well as average round-trip jitter were calculated.

As reported in [3], small audio packets of the same size evenly distributed in time at 30 milliseconds interval were used. In this research, case large size 1 KB packets at a shorter time interval of 12.5 milliseconds were used to observe inter-cloud quality of service characteristics. As stated in [32],Cisco's guideline on QoS of networks for the high quality demanding VoIP application requires networks to sustain at most 300 milliseconds round-trip latency, and average round-trip jitter of lower than 60 milliseconds, with packet-loss rate of less than 1%. The EC2 US-East/EU-West experimental results are summarized in **Table** 5.

As compared to the FutureGrid multi-meeting scenario discussed in Section 4.1.8.5.1.2 above in which a total of 150 meetings, each with 20 participants, shows an average round-trip latency below 50 milliseconds, the EC2 US-East/EU-West experiments simulated 12 meetings, each with 200 participants, showed a higher average round-trip latency of slightly above 90 milliseconds, which is still sufficiently below the maximum 300 milliseconds prescribed on Cisco network guideline for VoIP applications.

Table 5. Amazon EC2 Inter-cloud Quality of Service

Total Users	Min. RTT (milliseconds)	Max. RTT (milliseconds)	Average RTT (milliseconds)	Average Jitter (milliseconds)
200	90.15	124	99.51	16.70
400	91.09	133.81	108.38	26.92
600	90.61	155.79	109.80	28.67
800	91.21	183.69	107.56	29.67
1200	91.87	189.82	110.79	35.48
1400	92.18	165.74	106.39	38.69
1600	94.40	235.14	118.94	50.63
1800	93.56	197.89	110.80	33.77
2000	91.25	270.44	110.93	31.98
2200	108.30	318.08	151.66	74.33
2400	93.2	682.01	141.82	57.92

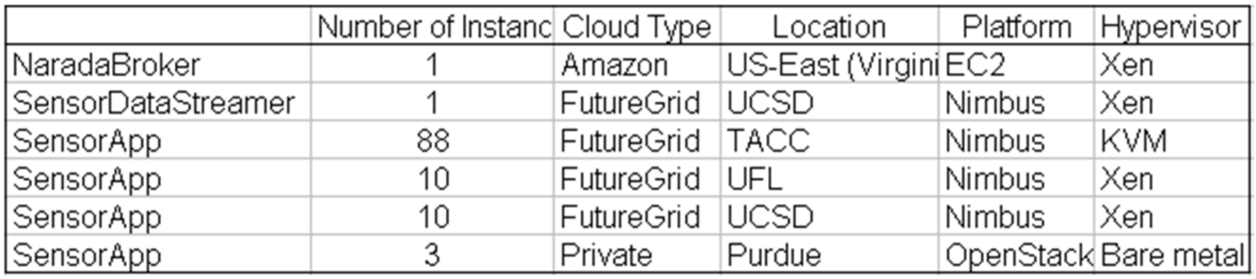
4.1.8.5.3 Hybrid Cloud – OpenStack, FutureGrid and Amazon EC2 Clouds

The study of hybrid cloud for sensor grid applications was preliminary. The focus was on understanding the viability and reliability of scaling up server resources for large-scale sensor deployment in different cloud regions that are operated by different providers using different cloud environments and technologies that encompass a combination of deployed private, community and public clouds; yet maintaining interoperability among the different services of an SCGMMS-type real-time, message-based sensor grid application in a heterogeneous and distributed cloud technology environment. An SCGMMS-type sensor grid application essentially boils down to independent message-capable service components interacting via messages. The current state of the interfaces and procedures that support launching virtual machines in cloud instances, and connecting to and monitoring individual instances is tedious and inconvenient even for the case of a single region by a single provider. The task to perform larger scale hybrid cloud experiments using heterogeneous environments across cloud providers and in different cloud regions or clusters will be even more tedious and cumbersome. For simplicity but without loss of generality, the research team modeled an SCGMMS-type sensor grid application by reducing it to a distributed, service-oriented application in which all the launched service components communicated via messages with representative message payload and frequency in an architecture that resembled an SCGMMS application.

The research team developed an NB application called SensorDataStreamer that streamed 256 bytes of data per second to an NB server. They also developed an NB application called SensorApp that consumed data streamed by SensorDataStreamer. It is worth noting that the objective in this set of experiments was not to exhaust the computing power in each instance but to understand the viability and reliability of scaling up and aggregating large on-demand computing resources in a distributed heterogeneous cloud environment to support large-scale SCGMMS applications. The particular choices of placement of NB, SensorDataStreamer and SensorApp on which type of cloud was not considered at this stage of the hybrid cloud research. An NB server was deployed in Amazon EC2 US-East (Virginia) public cloud. A SensorDataStreamer was deployed in an instance on FutureGrid Sierra community cloud (University of California, San Diego) using Nimbus. A Nimbus instance has computing power equivalent to a 2-core Intel Xeon X5570 with 12 GB RAM. The level of computing power of a Nimbus instance is similar to that of the EC2 instance reported in [10] which could support at least 2,400 VoIP application clients. Similar to the way SensorDataStreamer was deployed, each SensorApp was hosted in a separate cloud instance. A total of one hundred and eleven (111) cloud instances were launched for deploying SensorApp. One hundred and eight (108) of the 111 cloud instances were run on FutureGrid as a community cloud. The remaining three (3) instances on a private cloud using OpenStack. Among the 108 FutureGrid cloud instances, 88 were running at Alamo (Texas Advanced Computing Center, University of Texas) on Nimbus, 10 at Foxtrot (University of Florida) on Nimbus, and 10 at Sierra (University of California, San Diego). The OpenStack clouds were in Purdue.

A total of over 1.6 million 256-byte messages were communicated reliably over a duration of four hours among 112 distributed, heterogeneous cloud instances in a hybrid cloud setting. **Table 6** below summarizes the experimental setup.

Table 6. Hybrid Cloud Experimental Setup



	Number of Instanc	Cloud Type	Location	Platform	Hypervisor
NaradaBroker	1	Amazon	US-East (Virgini	EC2	Xen
SensorDataStreamer	1	FutureGrid	UCSD	Nimbus	Xen
SensorApp	88	FutureGrid	TACC	Nimbus	KVM
SensorApp	10	FutureGrid	UFL	Nimbus	Xen
SensorApp	10	FutureGrid	UCSD	Nimbus	Xen
SensorApp	3	Private	Purdue	OpenStack	Bare metal

4.2 Develop Enhanced Sensor Grid Application

While the Sensor Grid Middleware provides the fundamental capability for sharing sensor data after sensors are registered and begin publishing their data, applying this technology in a practical, operationally relevant application required the building and integration of some additional capabilities. This section of the report describes the R&D associated with building a set of clients and software components that use the Sensor Grid Middleware and provide an application that can be demonstrated and used in an operationally relevant scenario. During the early part of the research, the LVC SIDFOT program was identified as the focus for the development of an AMSA TO 4, Advanced Technology for Sensor Clouds research project application. To that end, the goal was to build a prototype application and demonstrate an enhanced Sensor Grid Application that would integrate individual sensors and provide a more complete picture of a designated environment with increased situational awareness that would be scalable and able to stand up to live and chaotic situations. Since the LVC SIDFOT program's target personnel of interest was a first responder, the Enhanced Sensor Grid Application software was developed with a first responder's mindset, and with the ability to be stood up on a remote site to help enable collaborators and first responders to more accurately and quickly assess the situation and maximize situational awareness.

The LVC SIDFOT program required the integration of live sensors with virtual and constructive sensors to provide a robust, large scale emergency response scenario in which the sensors (live, virtual and constructive) would provide situational awareness. The LVC SIDFOT concept is shown in **Figure 58**.

This figure shows that the LVC SIDFOT concept required the integration of live sensors with virtual and constructive sensors that were a part of a simulation or game environment. These virtual and constructive simulation constructs included multiple level fidelity simulations, training/rehearsal capability, simulations/game environments, viewers, Distributed Interactive Simulation (DIS) capable interactions, synthetic audio/images/videos, virtual/constructive simulation gateway(s), and the ability to provide connectivity and flexibility. These constructs are represented on the Virtual/Constructive side of Figure 58. The "Live" side of Figure 58 shows a representation of the live sensors located at the training site that are connected to the sensor grid. The "Live" side also represents the field demonstration/experimentation site, the instrument locations/activity at site (e.g., audio, video, human loc/status/activity), the communication mechanisms/devices, the digital/Internet connectivity, as well as the data collection and storage.

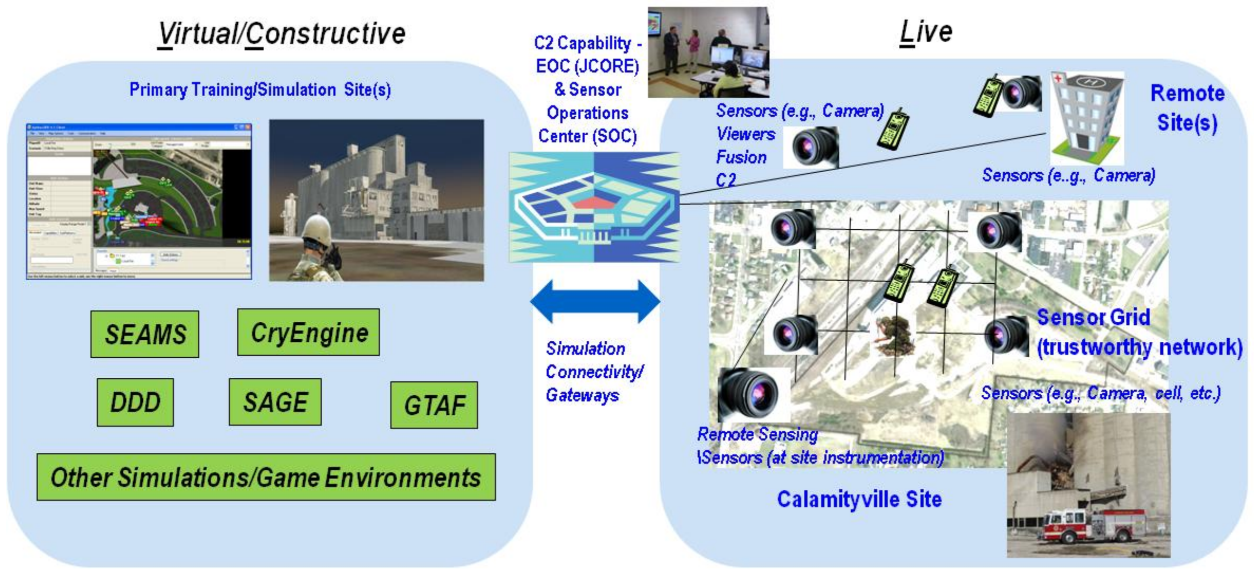

Virtual/Constructive
Primary Training/Simulation Site(s)
SEAMS
CryEngine
DDD
SAGE
GTAF
Other Simulations/Game Environments
C2 Capability - EOC (JCORE) & Sensor Operations Center (SOC)
Simulation Connectivity/ Gateways
Live
Sensors (e.g., Camera)
Viewers
Fusion
C2
Remote Site(s)
Sensors (e..g., Camera)
Sensor Grid (trustworthy network)
Sensors (e.g., Camera, cell, etc.)
Remote Sensing \Sensors (at site instrumentation)
Calamityville Site

Figure 58. LVC SIDFOT Concept

The R&D required for supporting the LVC SIDFOT project encompassed building software that worked with the Sensor Grid Middleware to support the integration of live, virtual and constructive sensors; as well as provide a capability to investigate how to access and report trust and trustworthiness associated with sensor grids.

The research team started with the basic Sensor Grid Middleware, an overview of which is illustrated in **Figure 59**. The Sensor Grid is the entire network of interconnected Sensor Grid nodes, the attached sensors "publishing" to the grid, and the listening clients "subscribing" to the sensors that are connected to the grid to support an operational scenario, the LVC SIDFOT scenario in particular. The Enhanced Sensor Grid Application consists of the Sensor Grid Testbed, which in turn consists of Sensor Grid Nodes, Sensors, Clients, and Operators. These will be described in the following paragraphs.

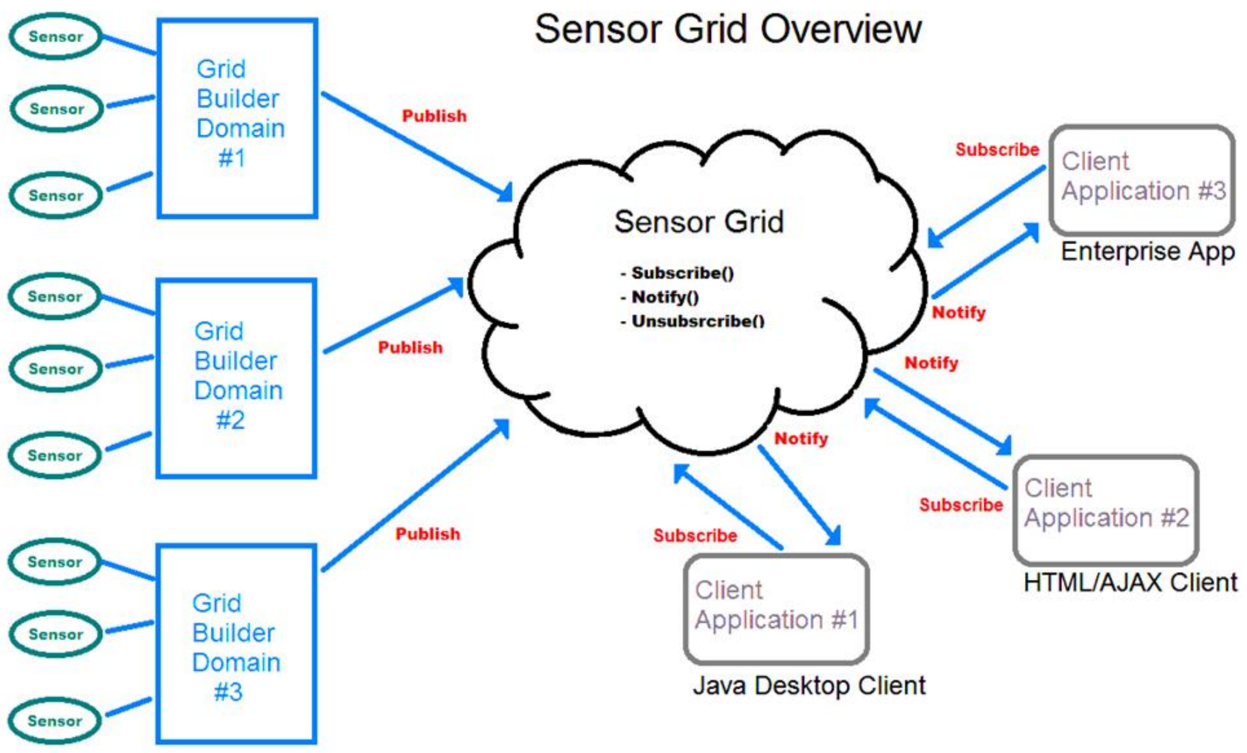


Figure 59. Sensor Grid Overview Map

4.2.1 Sensor Grid Testbed

The basic Sensor Grid Middleware software provides the interface and communication venue for sensors to share sensor data, but the basic software did not provide the fundamental tools to support researching trust and trustworthiness technologies for multi-layered sensor grids. To address this need, the research team decided to research, develop and integrate the tools to build a Sensor Grid testbed.

The purpose of the Sensor Grid testbed is to provide a researcher-friendly environment that enables and accommodates R&D of trust and trustworthiness concepts for multi-layered sensor grids. The tools developed for the Sensor Grid testbed, built on top of the Sensor Grid Middleware software package, are geared towards letting the researcher design and set up a network unique to their own multi-layered sensor requirements, and can be modified to provide functionality in various research testing scenarios. The Sensor Grid testbed is made up of several components providing different capabilities required for trust and trustworthiness research/testing, and can be modified individually.

The Sensor Grid testbed supports the establishment and instantiation of sensor grid “nodes” or “servers,” which run the Sensor Grid middleware software; a collection of sensor grid “sensors” which send data to the sensor grid nodes; and the sensor grid “clients” which receive data from the sensor grid nodes passed to them from the sensor grid sensors. The unique aspect of the Sensor Grid testbed is the inclusion of two special clients that support the archival and real-time rendering of the sensor data. The two special clients that the research team deemed necessary to support sensor grid research are the SensorGrid Archive Collector (SACK) and the SensorGrid Client for Operational Awareness (SCOPE). These two clients are depicted as they interface to the Sensor Grid in **Figure 60**.

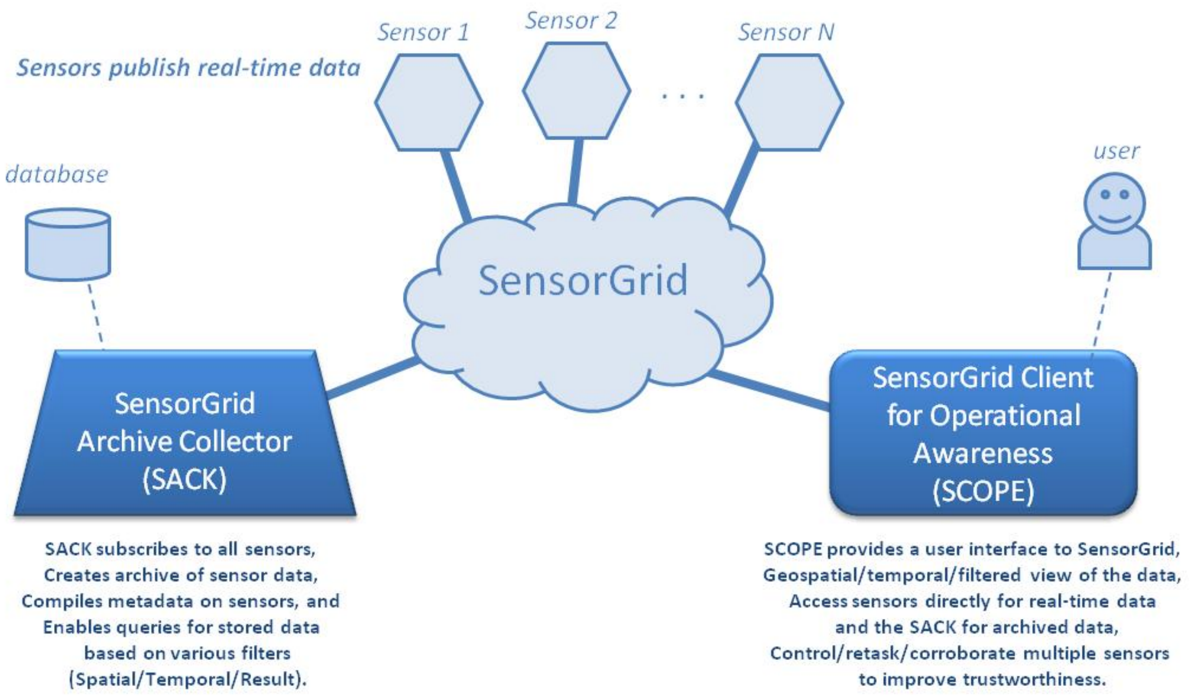
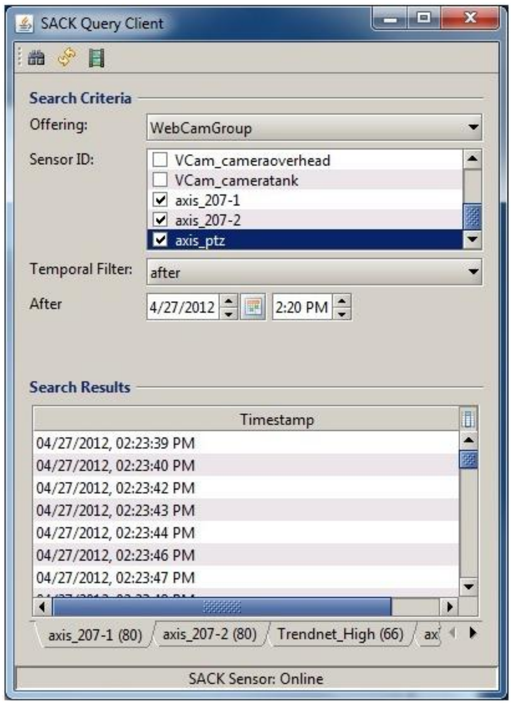

Sensor 1
Sensor 2
Sensor N
Sensors publish real-time data
database
user
SensorGrid
SensorGrid Archive Collector (SACK)
SensorGrid Client for Operational Awareness (SCOPE)
SACK subscribes to all sensors, Creates archive of sensor data, Compiles metadata on sensors, and Enables queries for stored data based on various filters (Spatial/Temporal/Result).
SCOPE provides a user interface to SensorGrid, Geospatial/temporal/filtered view of the data, Access sensors directly for real-time data and the SACK for archived data, Control/retask/corroborate multiple sensors to improve trustworthiness.

Figure 60. Sensor Grid Architecture with SACK and SCOPE

The SACK interfaces to the Sensor Grid Middleware to collect data from the registered sensors and store the data from the sensors in a database for future retrieval. The SCOPE interfaces to the Sensor Grid Middleware to provide a real-time rendering of sensor data to sensor grid users. The SCOPE’s real-time rendering shows the geospatial location of sensors, along with pertinent information about the sensors and a user friendly view of the sensor data. Additional design details for each of these components are provided in the following paragraphs.

4.2.2 SensorGrid Archive Collector (SACK)

A Sensor Grid data client, termed “the SACK,” was developed as an interface for data persistence, replay, and analytics. The SACK subscribes to all data being published by sensors on the sensor grid and thus serves as a data sink for the Sensor Grid. The SACK uses a relational database for the archival storage of the sensor data. The startup screen interface for the SACK is shown in **Figure 61**. This SACK startup screen shows a Java graphical user interface (GUI) that implements replay and query capabilities that were developed and serve as a proof of concept for future data exploitation tools.



Offering: WebCamGroup

Sensor ID:
- [ ] VCam_cameraoverhead
- [ ] VCam_cameratank
- [✓] axis_207-1
- [✓] axis_207-2
- [✓] axis_ptz

Temporal Filter: after

After: 4/27/2012 2:20 PM

Timestamp
04/27/2012, 02:23:39 PM
04/27/2012, 02:23:40 PM
04/27/2012, 02:23:42 PM
04/27/2012, 02:23:43 PM
04/27/2012, 02:23:44 PM
04/27/2012, 02:23:46 PM
04/27/2012, 02:23:47 PM

axis_207-1 (80) | axis_207-2 (80) | Trendnet_High (66) | ax

Figure 61. SACK Interface Diagram

A Java Graphical User Interface shown as the Observation Animator screen in **Figure 62** implements replay. The Observation Animator, coupled with the query capability shown in Figure 6161 above, was developed and serves as a proof of concept for future sensor data exploitation tools. The Observation Animator interface shown in Figure 6262 was developed to display time correlated SACK records. Temporal filtering may be applied by the user to restrict the dataset for observation. The small, sliding blue time scroll button located on the top of Figure 6262 permits the user to scroll either forward or backward in time while keeping the images from the four sensor data panels synchronized and time correlated.

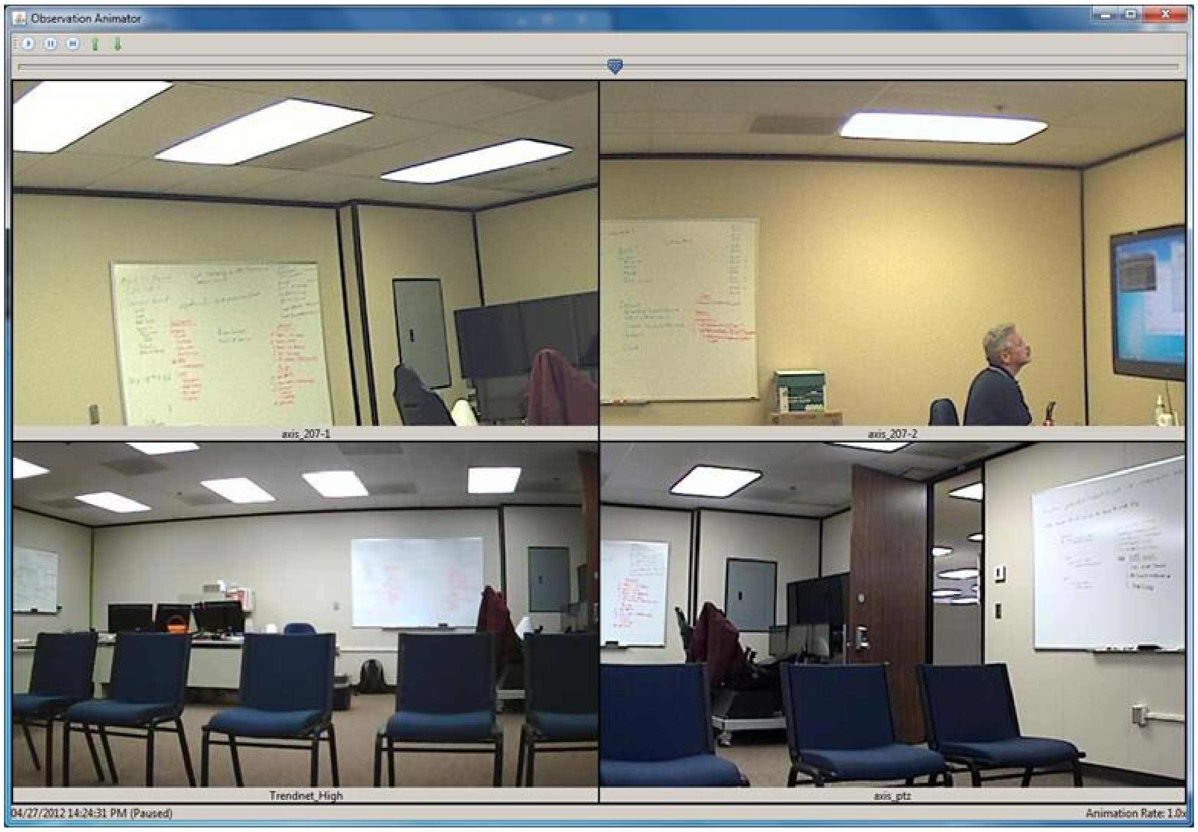


Figure 62. SACK Observation Animator Screen

The SACK interfaces directly to the Sensor Observation Service (SOS) and serves as the broker between the Sensor Grid and the Sensor Observation Service. The SACK is directly responsible for the integration between the SACK and the SOS operators. Integration to the SOS is significant because the SOS supports industry standard formats, protocols, and ontologies. Support for the SOS allows for the direct utilization of a variety of industry tools and algorithms. The SACK's sensor data storage interface between the Sensor Grid and SOS is depicted in **Figure 63**. When a query is issued for archived data, the SACK services the query by interfacing with the SOS. The SACK's sensor data access interface to provide for user friendly sensor data retrieval between the Sensor Grid and SOS is depicted in **Figure 64**.

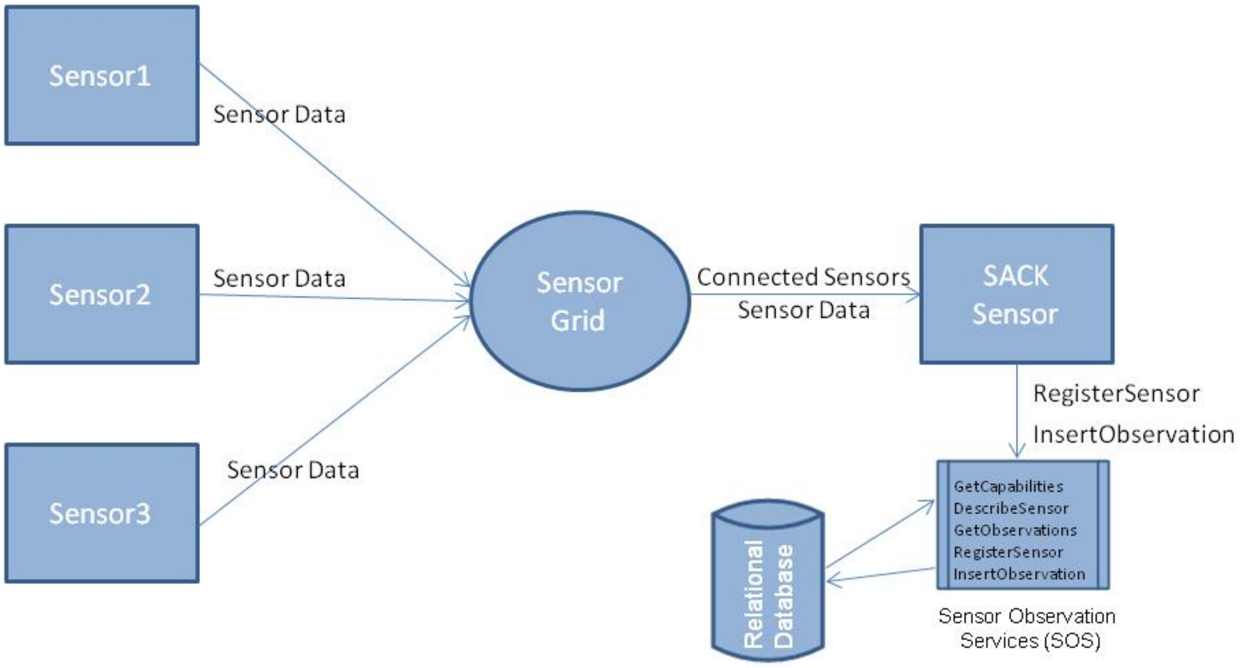


Figure 63. SACK Sensor Grid Data Storage Interface

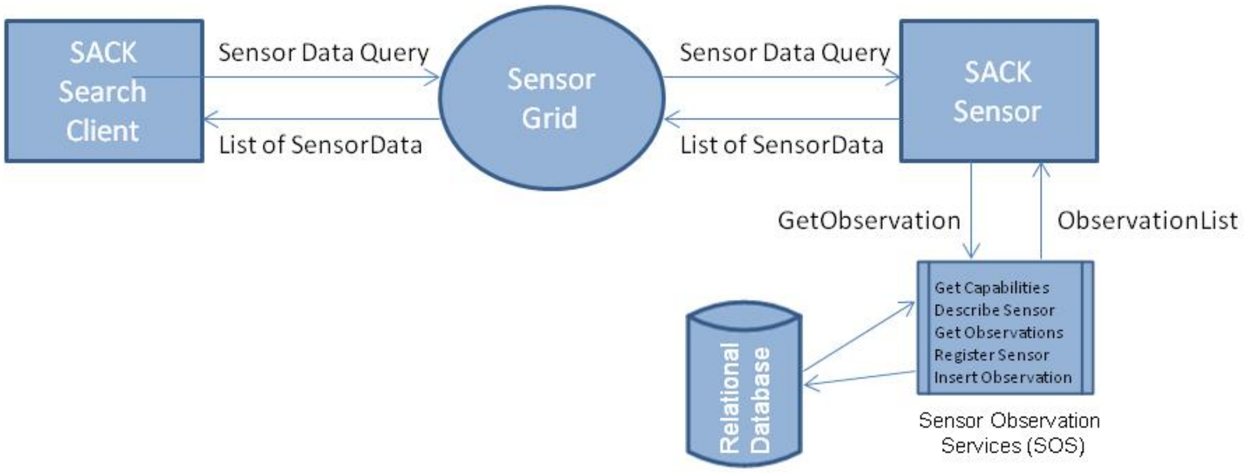


Figure 64. SACK Sensor Data Retrieval Interface

4.2.3 SensorGrid Client for Operational Awareness (SCOPE)

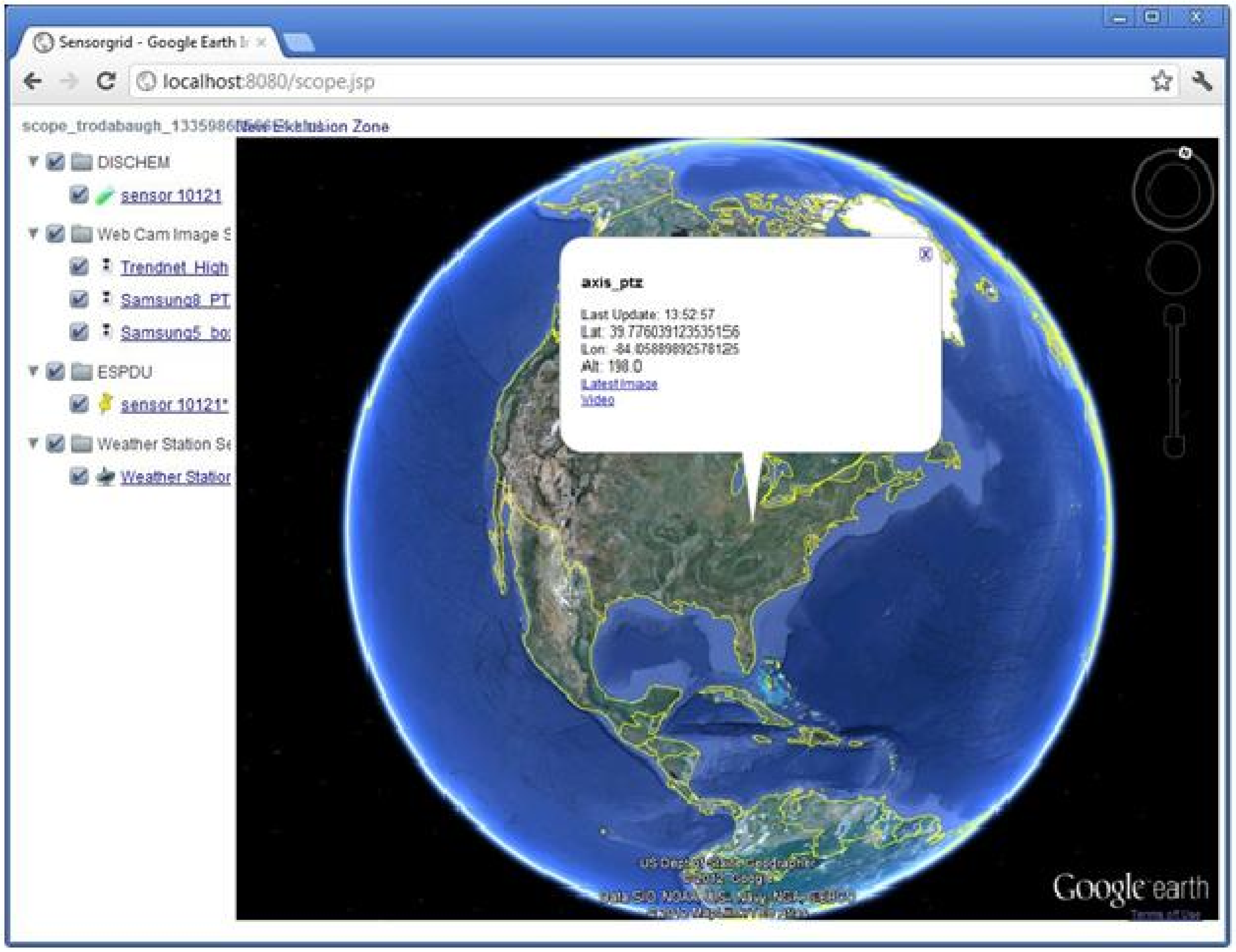
SCOPE, the other key client developed to support this research effort, is a situational awareness tool for the sensor grid. It provides a common, yet customizable view of each sensor on the grid. The SCOPE provides sensor grid users with quick insight into the state of all sensors on the grid. It is architected into two main components: a back end that generates Keyhole Markup Language

(KML) representations of sensors on the grid and a Google-earth based front-end for viewing the KML representations generated by the back end.

The SCOPE back-end runs within the context of a web application server such as Apache Tomcat. It consists of a series of “SCOPE Collectors” that generate KML representations of all sensors that a particular user wishes to view in the SCOPE client. SCOPE Collectors for each logged-in user are managed by a “SCOPE Manager”, and are each instantiated with a username, password and a set of filter criteria (e.g., a geospatial bounding area, sensor types, etc.). Each Collector becomes a sensor grid client, logged in with the specified username and password so that the user it is designated for only sees sensors that are permitted for that authenticated user. As it recognizes new sensors on the grid or data updates from existing registered sensors, it will update the appropriate KML object for that sensor.

The KML generated by these clients consists of a top-level KML file that has links to other KML files for each sensor type. Each link in this top-level file is set to refresh the contents of the linked files at a defined frequency so that clients will periodically refresh their views to reflect updates made to all of the KML objects by the SCOPE collector. In order for each type of sensor to be presented in the KML in customizable ways, the “Sensor Properties Generator” plugins can be written for each sensor type. Such plugins return a set of properties for a given sensor (i.e, its name, coordinates, icon, etc.) that are used to generate a KML object and require only the implementation of a Java interface containing a single method.

The SCOPE front-end is presented within a web page. Before gaining access to this web page, a login page is displayed to receive the user’s username and password. Upon successful authentication, a web page with three sections is displayed as shown in the **Figure 65** screenshot below. In the left section of the page is a tree containing all of the sensors meeting the user’s filter criteria and to which that user has permission to see. This tree is automatically updated with Java script to represent the latest state of the sensor grid. On the right side of the page is the Google Earth display that will show the relative geospatial location for all of the KML objects created by the SCOPE Collector for the user’s session. Since, as mentioned above, the top-level KML contains refreshable links to other KML files containing all the actual sensor definitions, the Google Earth display will dynamically update the locations and display of each sensor to represent their latest state within the sensor grid. Clicking on any object in the tree will “zoom in” the Google Earth display to the data for that sensor. Clicking on any sensor within the Google Earth display will display a pop-up with information generated from that type of sensor’s plugin described above. Some of these pop-ups may have links to more detailed views of that sensor’s data (e.g., webcams have a link to live video feeds). At the top of the page is a series of links to perform various actions such as defining a new region to display, change the filter criteria, or logging out of the client.



axis_ptz

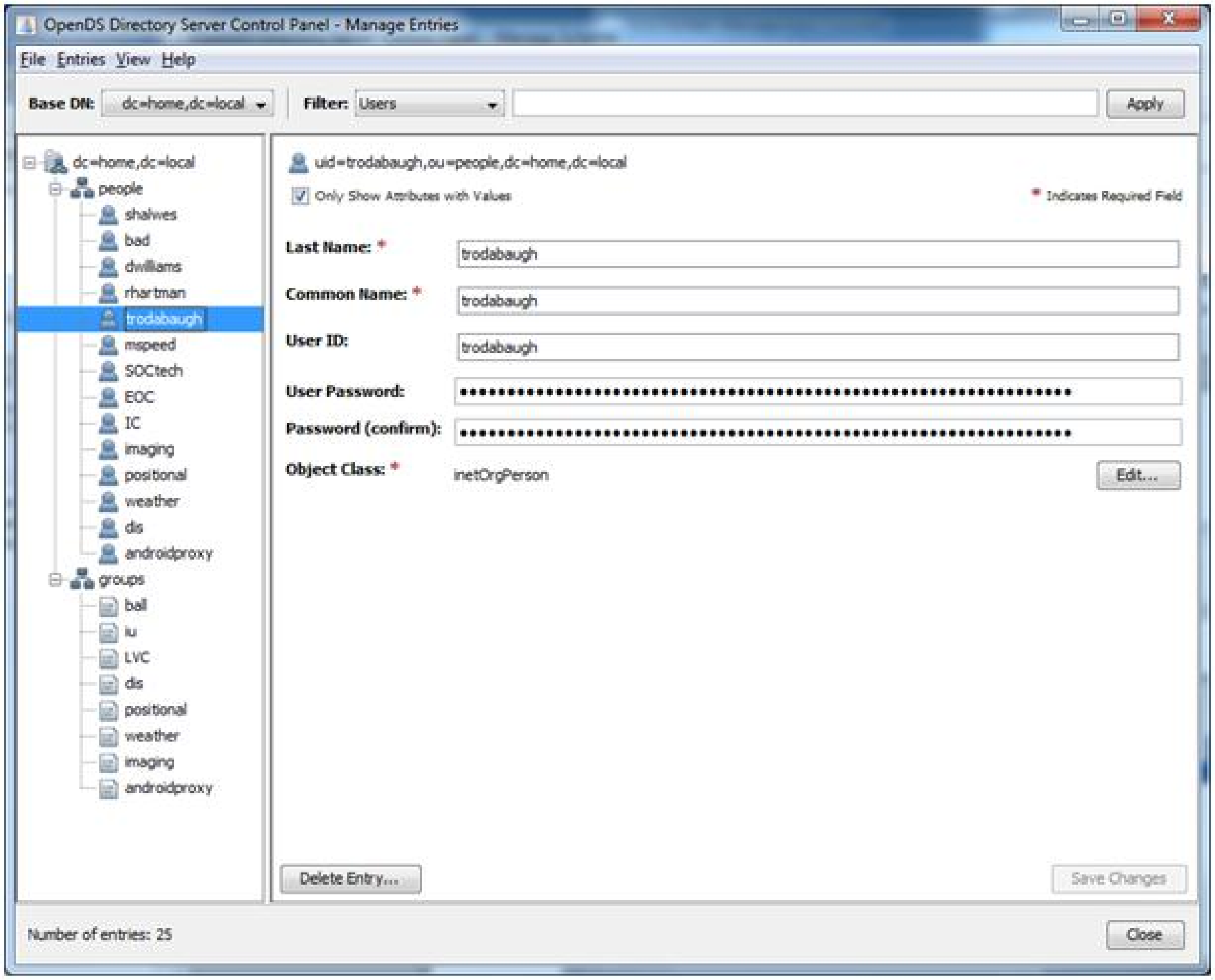
Last Update: 13:52:57
Lat: 39.776039123535156
Lon: -84.05889892578125
Alt: 198.0
Latest Image
Video

Figure 65. SCOPE Client Screenshot

While the SCOPE interface is currently used mainly to provide for sensor grid situational awareness, it is also envisioned to potentially provide a command and control interface. For example, clicking on a sensor could provide options to control a sensor’s connection to the grid or to provide links to interfaces for sending control commands to the sensor.

4.2.4 Lightweight Directory Access Protocol (LDAP)

The purpose of Authentication is to limit an unauthorized users access to critical sensor data, and restrict that user to only be able to view or edit authenticated sensor data. Lightweight Directory Access Protocol (LDAP) was chosen as the means of authentication due to its widespread use in the computing world today, as well as its very powerful and robust libraries and integration with a vast number of software suites. Being open source, LDAP is widely accepted as being the best free solution, with many different implementations, depending on the need. Indiana University successfully integrated LDAP with the Sensor Grid Middleware software and the Ball research team independently tested it thoroughly. Everyone who has an active LDAP account is "authenticated." Thus the distinction between unauthenticated and authenticated is unnecessary because those not listed are inherently and explicitly unauthorized. The LDAP Schema Tree is shown in **Figure 66**. Authenticated, on the other hand, is different than authorized. Authenticated means credentials (username and password) were cleared through the LDAP system. Authorized means one is able to access a given resource (e.g., someone in the imaging group can access web cams).



Base DN: dc=home,dc=local
Filter: Users

```
dc=home,dc=local
  people
    shalwes
    bad
    dwilliams
    rhartman
    trodabaugh
    mspeed
    SOCtech
    EOC
    IC
    imaging
    positional
    weather
    dis
    androidproxy
  groups
    ball
    iu
    LVC
    dis
    positional
    weather
    imaging
    androidproxy
```

uid=trodabaugh,ou=people,dc=home,dc=local

☑ Only Show Attributes with Values

* Indicates Required Field

Last Name: *	trodabaugh
Common Name: *	trodabaugh
User ID:	trodabaugh
User Password:	••
Password (confirm):	••
Object Class: *	inetOrgPerson

Number of entries: 25

Figure 66. LDAP Schema Tree

As seen in Figure 66 above, the LDAP schema has two main schema segments, “groups” and ”people.” Within the people segment (upper half of the left side of Figure 6666) are listed the end-users who possess accounts and passwords. And the groups segment (lower half of the left side of Figure 66) represents logical groupings of people.

There is no folder hierarchy within the “people” or “groups” segment of the tree. Within the “groups” segment, there are all the groups needed to specify what type of sensor or what amount of authority will be given to specific peoples’ accounts within each group. While it is understood that LDAP supports setting up groups of groups, the research team was unsuccessful in implementing this feature.

As a new user logs in using their LDAP credentials and attempts to start up a client or sensor, the group they have designated their sensor to start in is checked against their LDAP username to see if they have access to that group. If they have access, they are allowed to proceed and start the client or sensor, and register it with the Sensor Grid. If they do not have access permission, then access will fail and they are not allowed to start the software. Once a user is logged in, their sensors or clients can only see other sensors or clients in the groups that the user has access to. For instance, given a user in group A and group B, that user would only be able to see clients or sensors if the clients or sensors are either in group A, group B, or anonymous (unauthenticated) groups.

Sensors, when provisioned, set their role or group. Credentials must be passed when the sensor is instantiated that satisfy the set forth role or group. For example, consider the user “trodabaugh” in Figure 66 along with a sensor that specifies a role of “imaging.” When the sensor is instantiated, the credentials for “trodabaugh” are used. If “trodabaugh” is in the “imaging” group, then the sensor is instantiated. If “trodabaugh” is not in the “imaging” group, the sensor is not permitted to join the sensor grid

Clients, when provisioned, specify credentials (not a group) in the form of a username and a password. And any sensors allowed to be used with the specified credentials will be visible to the client. Anonymous sensors can be instantiated. Anonymous clients can be instantiated. Anonymous client see only anonymous sensors.

4.3 Research and Implementation of Trustworthiness Algorithms

As noted in Section 3.2.3 above, part of this research focused on investigating supporting technologies associated with sensor trust and trustworthiness. The research team worked with AFRL personnel to identify areas of research related to development of possible collective trust algorithms. Specifically, this research team extended previous research efforts associated with a database of trust metrics and analysis services for current and projected trust estimates and looked at applying these technologies to sensor grid cloud architectures.

Later on during the second year of the research effort, the research team identified some services to enhance trust and ensure a higher level of security for the sensor grid middleware. Additionally, the team developed several sensor vulnerability vignettes that were could be used with the sensor grid testbed for use in operational applications. The Secure Cloud Computing and Sensor Vulnerability Vignettes will be discussed in the following sections.

4.3.1 Secure Cloud Computing

The AMSA TO 4 research team explored how to implement secure cloud computing while operating on a brokered, trusted sensor network. The team investigated a model for large-scale smartphone based sensor networks, with sensor information processed by clouds and grids, where a mediation layer accomplished processing and filtering via a brokering network. In this proposed scenario, they assumed that aggregate results are sent to users through traditional cloud interfaces such as browsers. They conjectured that such a network configuration would include significant sensing applications. As part of this research they performed some preliminary system definition and considered threats to the system. Then, the core part of their research focused on solving three portions of the overall security architecture: i) Risk Analysis relating to the possession and environment of the smart-phone sensors, ii) New malware threats and defenses installed on the sensor network proper, and iii) An analysis of covert channels being used to circumvent encryption in the user/cloud interface.

As a result of this part of the research, the AMSA TO 4 research team outlined a high-level architecture that should both be realizable, and provide for the ability to perform on- demand analysis and processing of data from a large number of heterogeneous and globally placed sensors. The network is structured so that it is feasible to consider real or near-real time processing and interpretation of the data with appropriate resources. [They determined that there are still challenges with determining how to assure privacy, integrity and provenance of the data from its collection, through its life-cycle of processing to final consumption.] The research concluded that the largest research questions based on their architecture model lie at the tail ends of the data life-cycle. They identified several open research questions associated with data-collection by smartphone sensors and in the final delivery of a processed data-consumable. Their research identified specific directions aimed at solving these problems which are summarized in **Table 7**. The entire paper related to the results of this part of the research, titled *Secure Cloud Computing with Brokered Trusted Sensor Networks*, is provided at **Appendix A**.

Table 7. Summary of the Three Threats, Associated Dangers and Mitigation Strategies.

Threat	**Danger**	**Mitigation**
Sensor Abduction.	Malicious Sensor Data.	Detection of Non-regular Usage.
Side-channel Information Leakage.	Communication encryption is circumvented by analysis of packet sizes and spacing.	Flow Analysis and Padding.
Sensor Malware.	Sensor Data Theft.	Sensor Access Control Models.

4.3.2 Sensor Vulnerability Vignettes

About mid-way through the research project timeline, the LVC SIDFOT research team performed a vulnerability analysis of the overall Sensor Grid middleware being hosted on the recommended server configuration. The complete results of this analysis is documented in the Warfighter Interface Research and Technology Operations (WIRTO) Task Order (TO) 40 LVC SIDFOT Spiral 1 Interim Report, which can be obtained through the 711th Human Performance Wing (HPW) Warfighter Interface Division (711th HPW/RHC). The key vulnerability that this research team identified as being critical to address was the lack of authentication for sensors associated with the sensor grid. In response to this identified shortfall, the AMSA TO 4 research team decided to add an authentication capability to the Sensor Grid middleware software. To provide an authentication capability, the LDAP feature was added. The LDAP capability was discussed in detail earlier in Section 4.2.4.

To further investigate the ability to perform sensor trust and trustworthiness concepts, the team researched, developed and implemented two cyber-related attacks on the Sensor Grid testbed. The first attack was a Denial of Service (DOS) attack. The second attack was a sensor spoofing attack. These two attacks were developed and executed to prototype the usefulness of the sensor grid testbed for future trust and trustworthiness research on sensors operating in a grid.

4.4 Prototype Development, Integration, and Demonstration

Much of the preceding discussion covers the overall concept for the Enhanced Sensor Grid Application, along with the architecture and interfaces for the Sensor Grid Testbed, the SACK and the SCOPE. Furthermore, the LDAP discussion identified the rationale for, and some implementation details for the user and sensor authentication scheme. The following sections will describe the development of the sensors or sensor interface software, and sensor clients.

4.4.1 Development Phase

4.4.1.1 Sensors

For the purpose of this report, in most all cases the term sensor is synonymous with sensor interface software. For every sensor, live or virtual, there is a need for software that registers the sensor and publishes the sensor data as it becomes available through the sensor hardware.

4.4.1.1.1 Weather Station Sensor

The purpose of the weather station sensor is to collect weather data from the location where it is deployed. The research team decided to incorporate a weather station sensor into the Enhanced Sensor Grid Application because it is a valuable sensor aiding first responders in emergency situations. The local weather conditions dictate how hazardous materials and chemicals are atmospherically dispersed in an emergency scenario. The research team acquired a weather station sensor (shown in **Figure 67**) and developed the appropriate interface software to connect the sensor to the sensor grid.

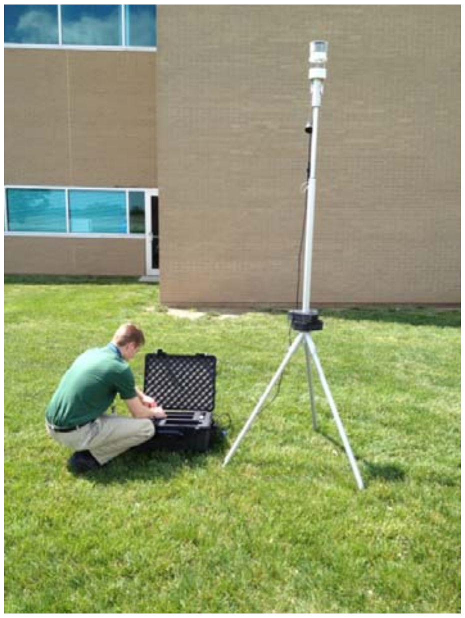


Figure 67. Weather Sensor on Tripod

The Weather Station Interface software interacts with the sensor grid middleware by registering the weather sensor with the sensor grid, querying the weather station and publishing the weather data to the sensor grid. The Weather Station Interface software queries the weather station periodically via a web service call to the Weather Station Microserver. The Weather Station Microserver is a small stand-alone networked computing device that logs all data from the weather station itself. The web service call returns XML containing the latest weather data readings from the weather station. The XML is parsed and published to the grid. The diagram shown in **Figure 68** illustrates how data is collected from the weather sensor and published to the sensor grid.

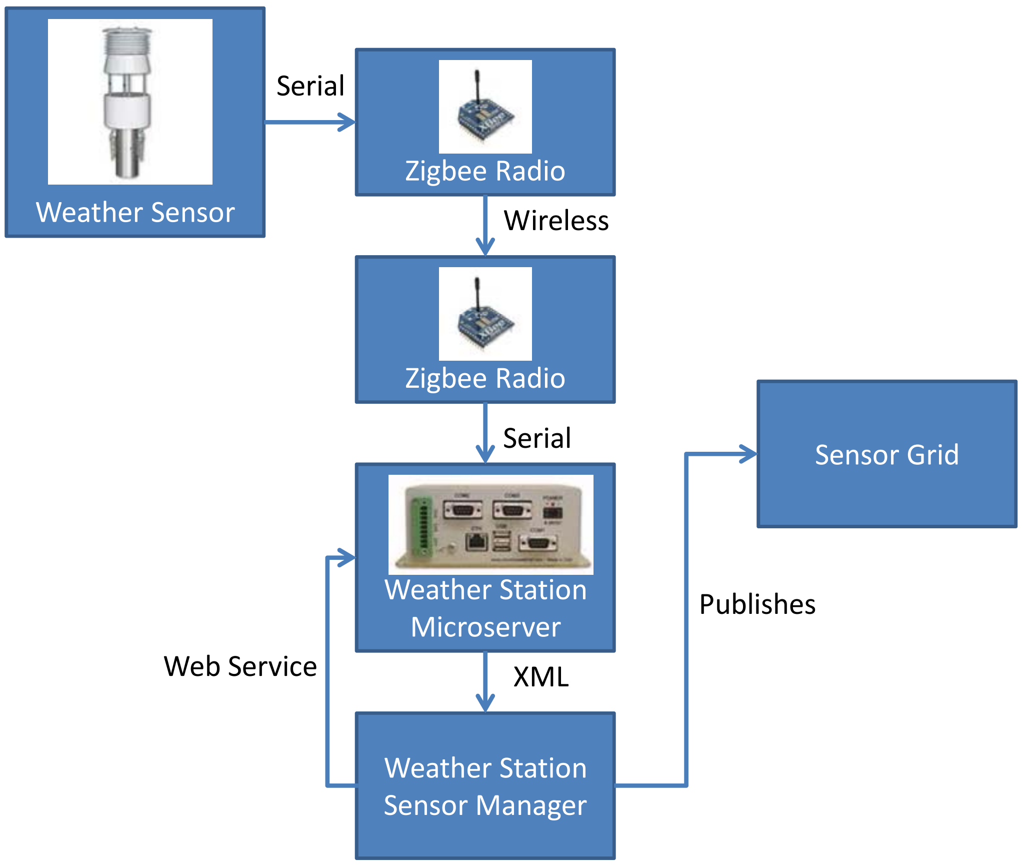
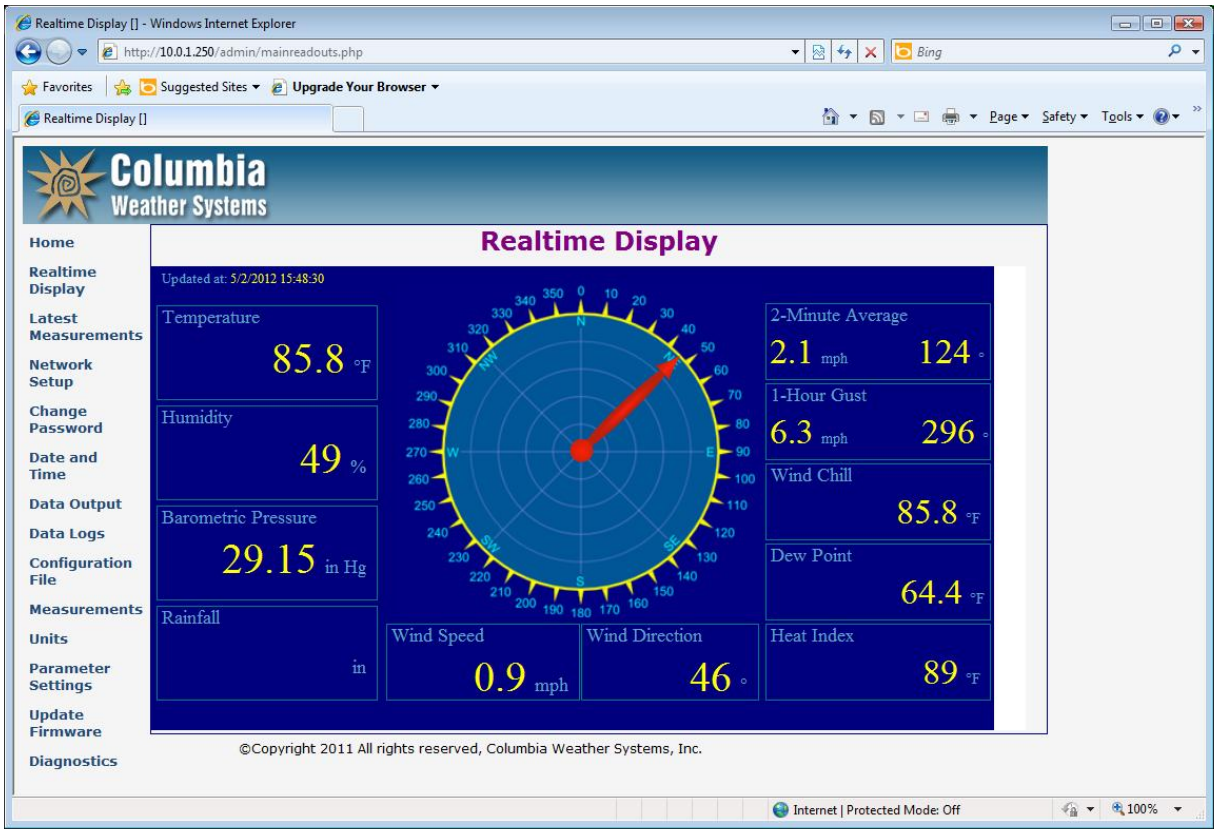


Figure 68. Weather Sensor Data Flow

While the weather data can be accessed directly without using the sensor grid by logging into the Weather Microserver (see **Figure 69**), the research team developed a Weather Sensor Client Data viewer for use with the SCOPE.



Realtime Display

Updated at: 5/2/2012 15:48:30

Temperature 85.8 °F

Humidity 49 %

Barometric Pressure 29.15 in Hg

Rainfall in

Wind Speed 0.9 mph

Wind Direction 46 °

2-Minute Average 2.1 mph 124 °

1-Hour Gust 6.3 mph 296 °

Wind Chill 85.8 °F

Dew Point 64.4 °F

Heat Index 89 °F

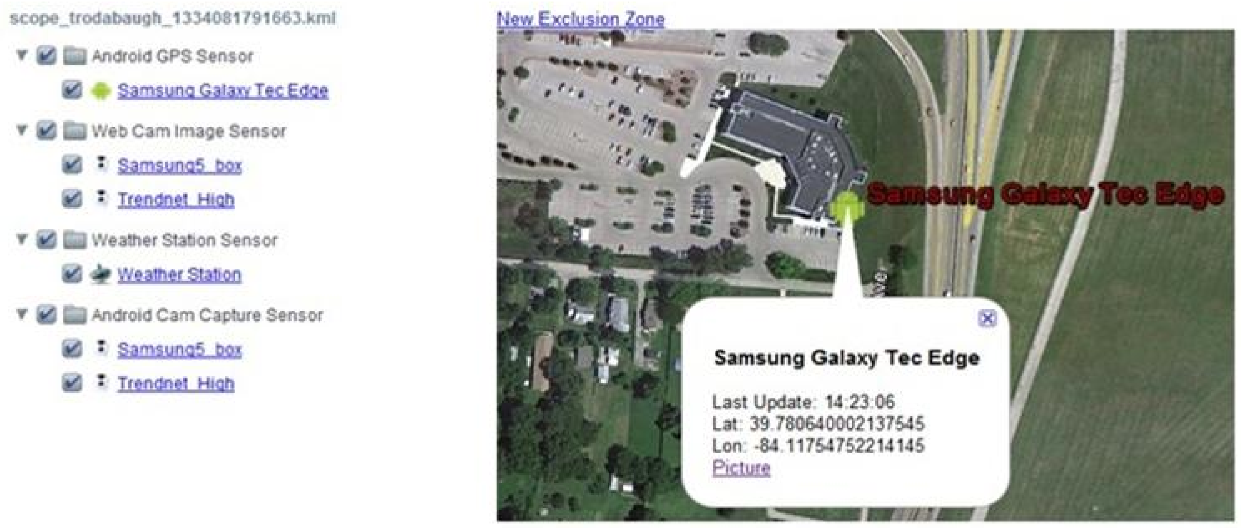
©Copyright 2011 All rights reserved, Columbia Weather Systems, Inc.

Figure 69. Web-based Columbia Weather System Realtime Display

4.4.1.1.2 GPS Sensor

While a GPS sensor provides latitude and longitude data associated with the actual location of the sensor, the purpose of the GPS Sensor software interface is to be able to read in data from a physical GPS device and to publish this data to the Sensor Grid. An overhead rendering of the GPS sensor location through the SCOPE, and associated GPS data is show in **Figure 70**. The callout window shows on the SCOPE display when the GPS sensor icon is selected by the mouse.

The data available from the GPS sensor will be in a specific format (NMEA) and will need to be accessible to the computer hosting the sensor in some fashion, such as bluetooth or USB. This data can also be supplied to the sensor software virtually when you start up the sensor program. When the data is supplied virtually it will update the output to the exact same location every second.



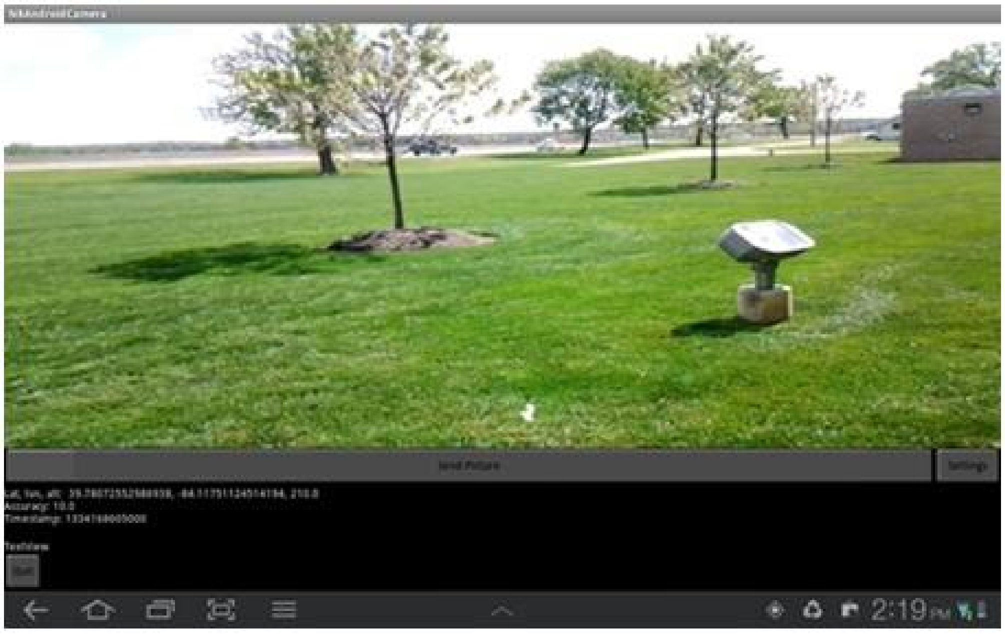
Samsung Galaxy Tec Edge

Last Update: 14:23:06
Lat: 39.780640002137545
Lon: -84.11754752214145
Picture

Figure 70. GPS Sensor Data View

4.4.1.1.3 AndroidCamCapture Sensor

The purpose of the AndroidCamCapture Sensor software interface is to support the publishing of pictures from an Android device to the sensor grid. The AndroidCamCapture Sensor software first registers the Android camera with the sensor grid, and then publishes the captured picture to the sensor grid. The rendering of data captured and published by an Android camera sensor is shown in **Figure 71**.



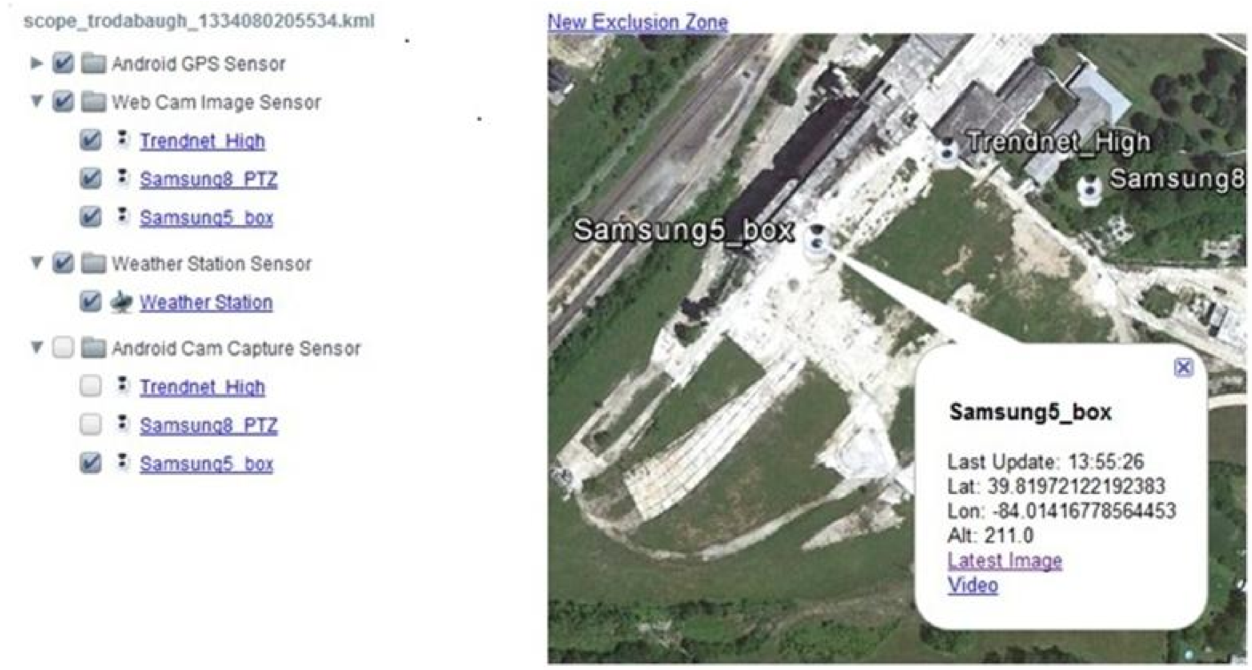
Lat, lon, alt: 39.78072552588938, -84.11751124514194, 210.0
Accuracy: 10.0
Timestamp: 1334168609000

Figure 71. Rendering of Android Camera Sensor Data

The AndroidCamCapture sensor is architecturally different from other sensor grid sensors. Since computing power and resources are limited on Android devices, Android devices must connect to the sensor grid at a much lower level via the Narada Brokering communication software. Thus, the AndroidCamCapture software serves not only as a traditional sensor grid sensor but also as a proxy which elevates published data from the Narada Broker level to the sensor grid level. The raw pictures bytes are received from the Android device and then repackaged and republished at the traditional sensor grid level.

4.4.1.1.4 IP Camera Sensor

An IP Camera Sensor provides video or Joint Photographic Experts Group (JPEG or JPG) streams from the place where they are located based on the direction they are pointing at. The purpose of the IP Camera sensor software is to be able to collect video or JPG streams from cameras on the local network or even on the Internet. The IP Camera Sensor software registers the sensor, then collects data from the sensor and publishes the sensor data to the sensor grid for use by sensor grid clients. An overhead rendering of an IP camera sensor location through the SCOPE, and associated location data is shown in **Figure 72**. Selecting the "latest Image" or "Video" link on the IP Camera Sensor data popup initiates a new window showing the selected data item.



Samsung5_box

Last Update: 13:55:26
Lat: 39.81972122192383
Lon: -84.01416778564453
Alt: 211.0
Latest Image
Video

Figure 72. SCOPE Rendering of IP Camera Sensor Data

The IP Camera sensor must be fed a URL to the direct JPG image, and then the sensor software will poll the URL at a specified rate, continually updating the latest JPG image. The sensor can also be set to only update on the user's request. Each image is then individually published to the Sensor Grid.

4.4.1.1.5 Mobile Platform Sensor

The purpose of the Mobile Platform sensor software is to collect data from the mobile platform hardware and publish it to the Sensor Grid. Its secondary purpose is to receive commands sent from the Mobile Platform client software and to execute them in hardware. A view of a mobile sensor platform is shown in **Figure 73**.

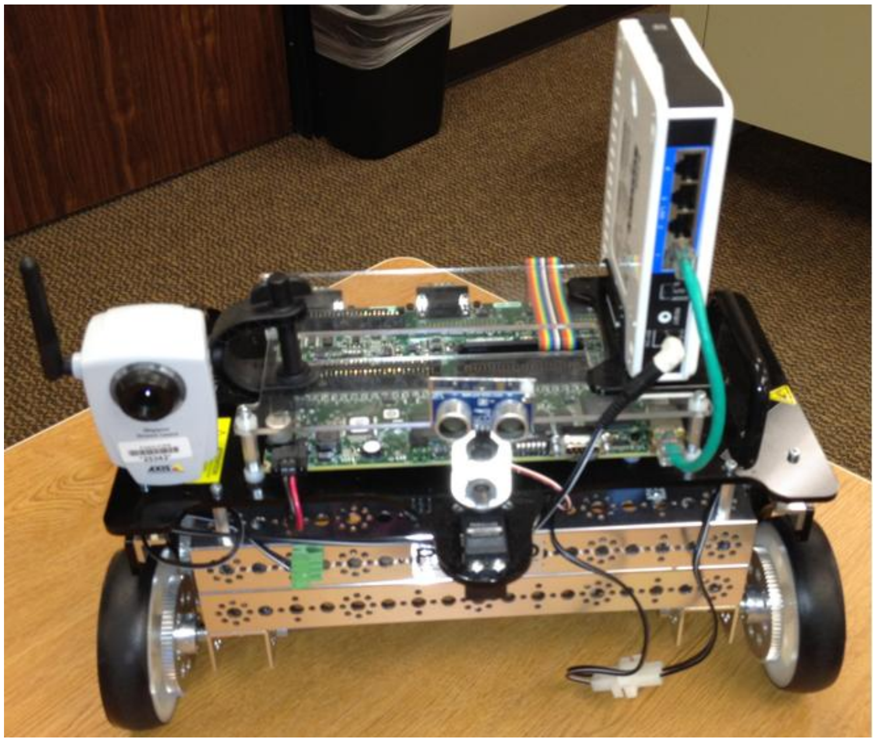
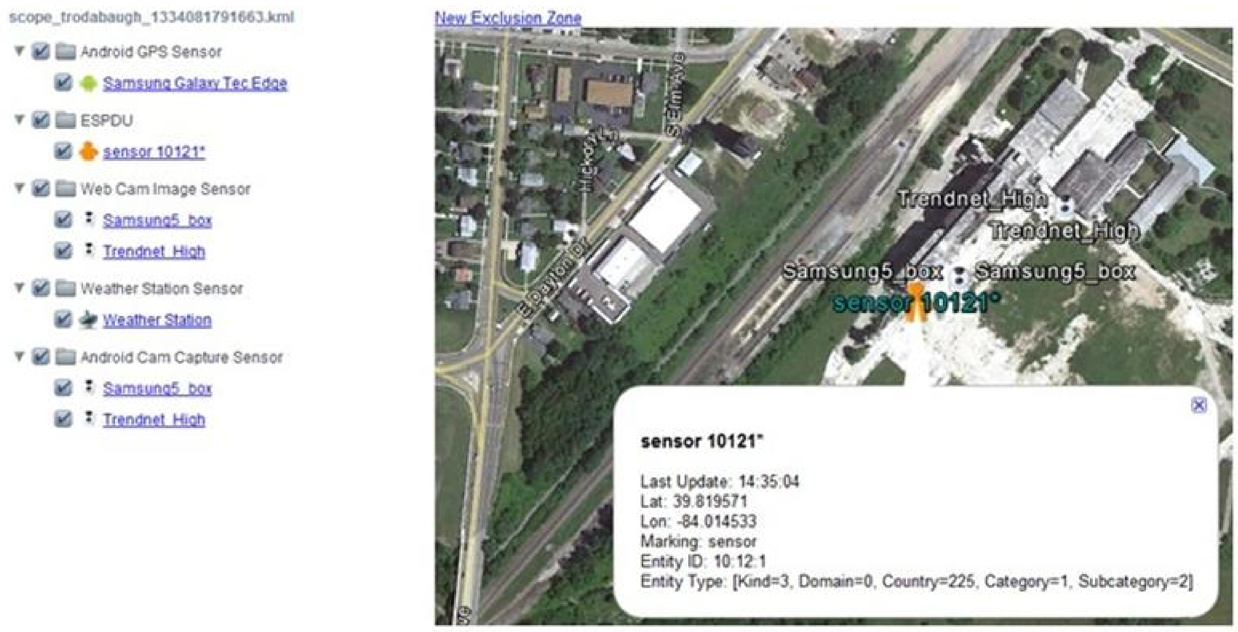


Figure 73. Mobile Platform Sensor View

The Mobile Platform sensor software exists as a web service on the mobile platform itself. The wireless router mounted on the top of the Mobile Platform allows it to roam around freely and send its data back to the client, as well as receive commands and execute them immediately. The mobile platform sensors shown in Figure 73 include an IP video webcam, a GPS sensor and a pair of oscillating mounted ultrasonic sensors.

4.4.1.1.6 DIS Sensor

The purpose of the Distributed Interactive Simulation (DIS) sensor is to collect state data from simulated entities, and publish the DIS state data through the sensor grid. The published DIS data is entity state data primarily associated with simulated entity movement. DIS is commonly used by the military and Department of Defense for training during Distributed Mission Operations Training (DMOT) exercises. A view of the SCOPE interface for a DIS sensor (simulated entity) is shown in Figure 74. In this case, the DIS data is latitude and longitude, as shown in the sensor popup window in the bottom right portion of **Figure 74**.

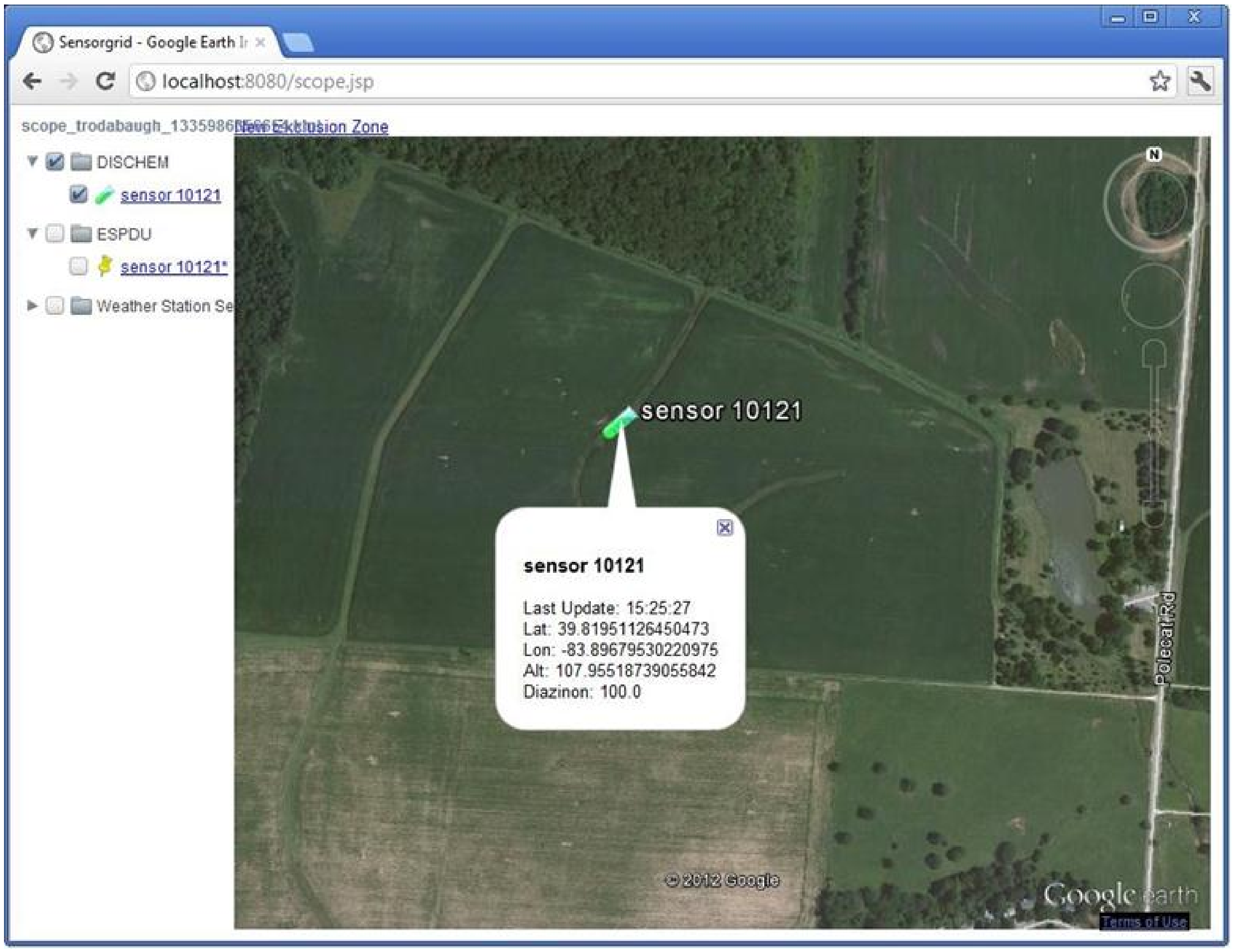


sensor 10121*

Last Update: 14:35:04
Lat: 39.819571
Lon: -84.014533
Marking: sensor
Entity ID: 10:12:1
Entity Type: [Kind=3, Domain=0, Country=225, Category=1, Subcategory=2]

Figure 74. SCOPE Screen Shot for DIS Sensor

The DIS sensor listens for and ingests DIS Entity State Protocol Data Unit (PDU) data. The Entity State PDU contains positional data (linear velocity and acceleration, latitude, longitude, angular velocity and orientation data) and allows for tracking of virtual and live DIS assets. The DIS sensor is capable of filtering the data in a variety of ways to ensure only data deemed interesting by the operator is passed through the sensor grid. The DIS sensor dynamically provisions and creates new sensors when a new asset is discovered. Similarly, if the entity is already provisioned as a sensor grid sensor, then the DIS sensor publishes relevant data through the already existing sensor. The location, DIS entity geographical data and the simulated sensor data for a simulated chemical sensor is shown in **Figure 75**.



sensor 10121

Last Update: 15:25:27
Lat: 39.81951126450473
Lon: -83.89679530220975
Alt: 107.95518739055842
Diazinon: 100.0

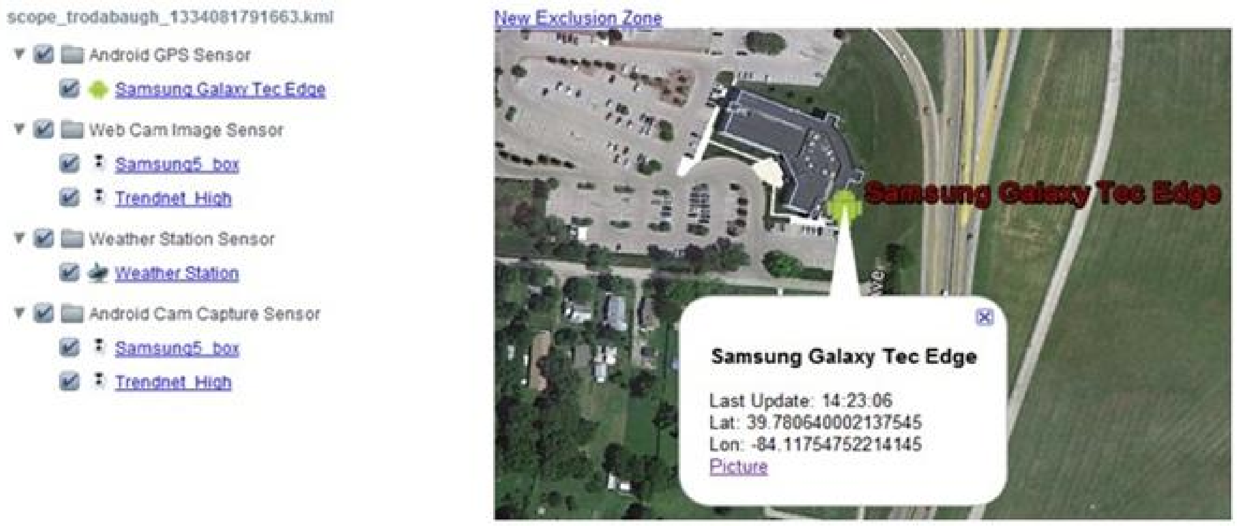
Figure 75. SCOPE View of Simulated Chemical Sensor

4.4.1.2 Clients

In the same manner that there are sensors and corresponding sensor interface software components, there are also sensors and corresponding sensor clients. However, in the case of sensor clients, the sensor client software registers with the sensor grid and subscribes to receive sensor data and then renders that data for sensor grid users.

4.4.1.2.1 Android Client

The purpose of the Android application is to track, using GPS, the application operator. Secondarily, the application allows for the capture of geotagged and annotated images. Both the images and the GPS data are published to the sensor grid. A view of the SCOPE interface for an Android client is shown in **Figure 76**.



Samsung Galaxy Tec Edge

Last Update: 14:23:06
Lat: 39.780640002137545
Lon: -84.11754752214145
Picture

Figure 76. SCOPE View of Android Client Data

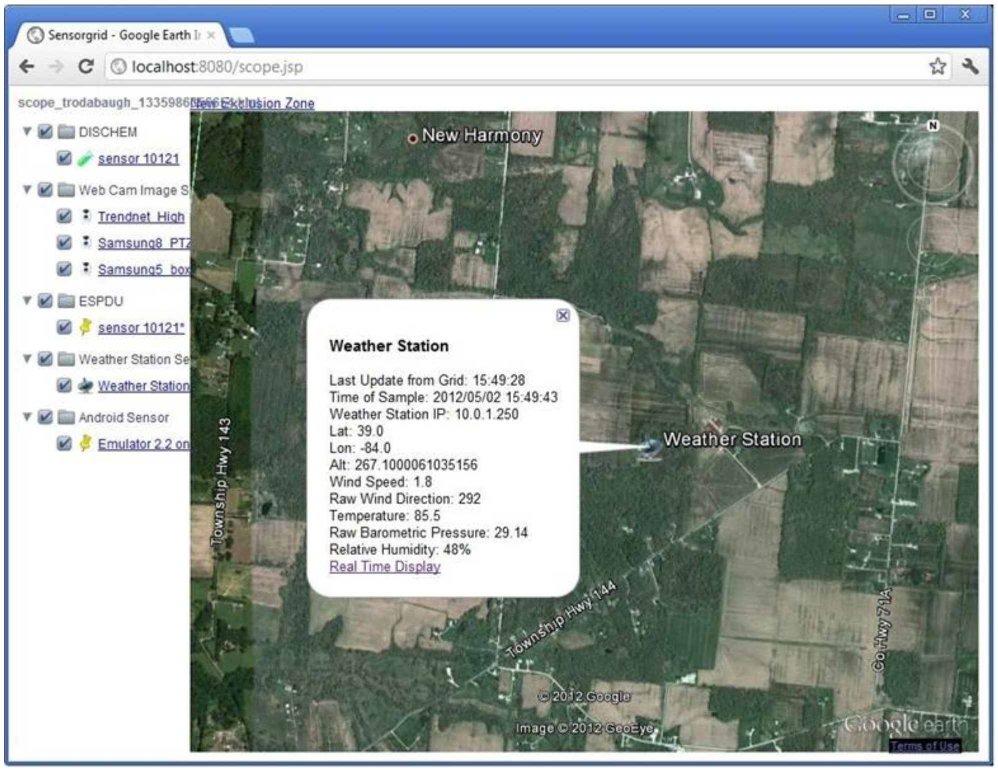
Once a GPS fix is obtained, the GPS information is continually published to the grid. Pictures can be taken on demand. If GPS is available, the pictures are geotagged with location information. An optional annotation is added to the image. All image metadata is embedded in the image file itself using Exchangeable image file format (or Exif) data. Connection settings and device names can be changed by way of a settings dialog.

4.4.1.2.2 Weather Station Client

The weather station client displays data from the weather station. The SCOPE provides a rendering that shows the geospatial location for the weather station and displays the data associated with the weather station. The SCOPE weather station client interface is shown in **Figure 77**.

The weather station client subscribes to weather station data. When weather station is received by the client, the data is parsed and displayed on the command line. The weather station client was completed merely as a proof of concept that the weather station sensor was operating correctly. The inclusion of the weather station in the Google Earth client outmoded the weather station client.

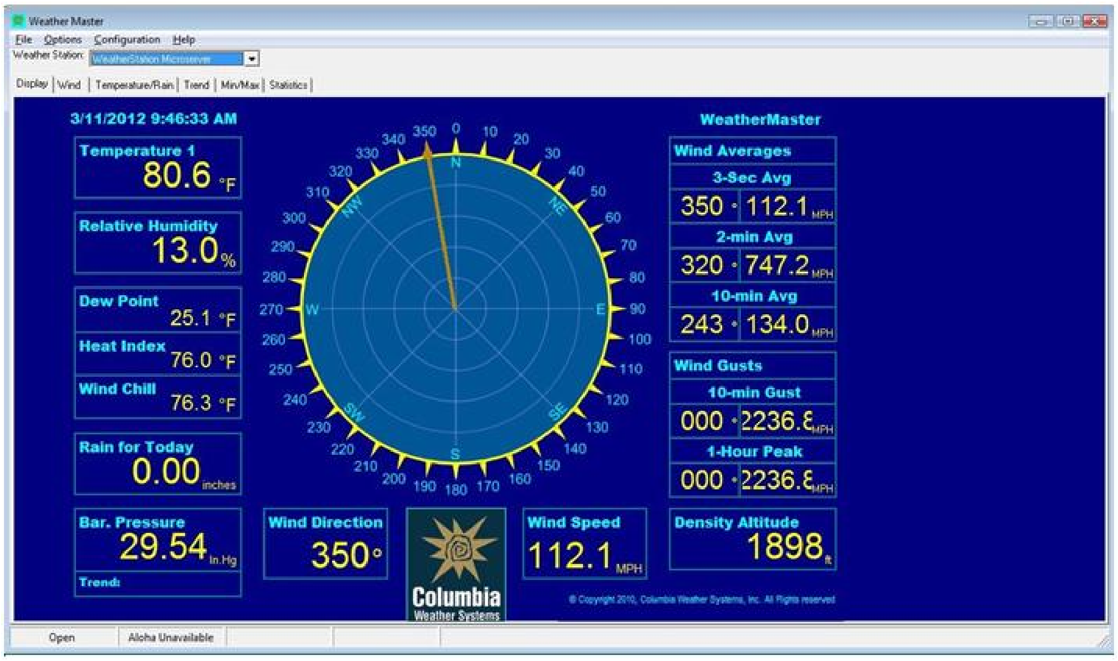
Processing the data published by the weather sensor allows for real-time weather data to be available through the sensor grid and be graphically depicted through a SCOPE window when the "Real Time Display" link is selected. A user friendly rendering of weather data is provided when this link is selected as shown in **Figure 78**.



Weather Station

Last Update from Grid: 15:49:28
Time of Sample: 2012/05/02 15:49:43
Weather Station IP: 10.0.1.250
Lat: 39.0
Lon: -84.0
Alt: 267.1000061035156
Wind Speed: 1.8
Raw Wind Direction: 292
Temperature: 85.5
Raw Barometric Pressure: 29.14
Relative Humidity: 48%
Real Time Display

Figure 77. SCOPE View for a Weather Station Client


Weather Master
File Options Configuration Help
Weather Station: WeatherStation Microserver
Display Wind Temperature/Rain Trend Min/Max Statistics
N
NE
E
SE
S
SW
W
NW
Columbia Weather Systems
© Copyright 2010, Columbia Weather Systems, Inc. All Rights reserved
Open
Aloha Unavailable

3/11/2012 9:46:33 AM

Temperature 1 80.6 °F

Relative Humidity 13.0 %

Dew Point 25.1 °F

Heat Index 76.0 °F

Wind Chill 76.3 °F

Rain for Today 0.00 inches

Bar. Pressure 29.54 In.Hg

Trend:

Wind Direction 350°

Wind Speed 112.1 MPH

WeatherMaster

Wind Averages

3-Sec Avg 350 • 112.1 MPH

2-min Avg 320 • 747.2 MPH

10-min Avg 243 • 134.0 MPH

Wind Gusts

10-min Gust 000 • 2236.8 MPH

1-Hour Peak 000 • 2236.8 MPH

Density Altitude 1898 ft

Figure 78. Real Time Rendering of Weather Sensor Data

4.4.1.2.3 Google Earth Client

The purpose of the Google Earth Client is to subscribe to GPS sensors' data and to output a KML file that may be viewed through Google Earth. The SCOPE Google Earth Client interface is shown in **Figure 79**.

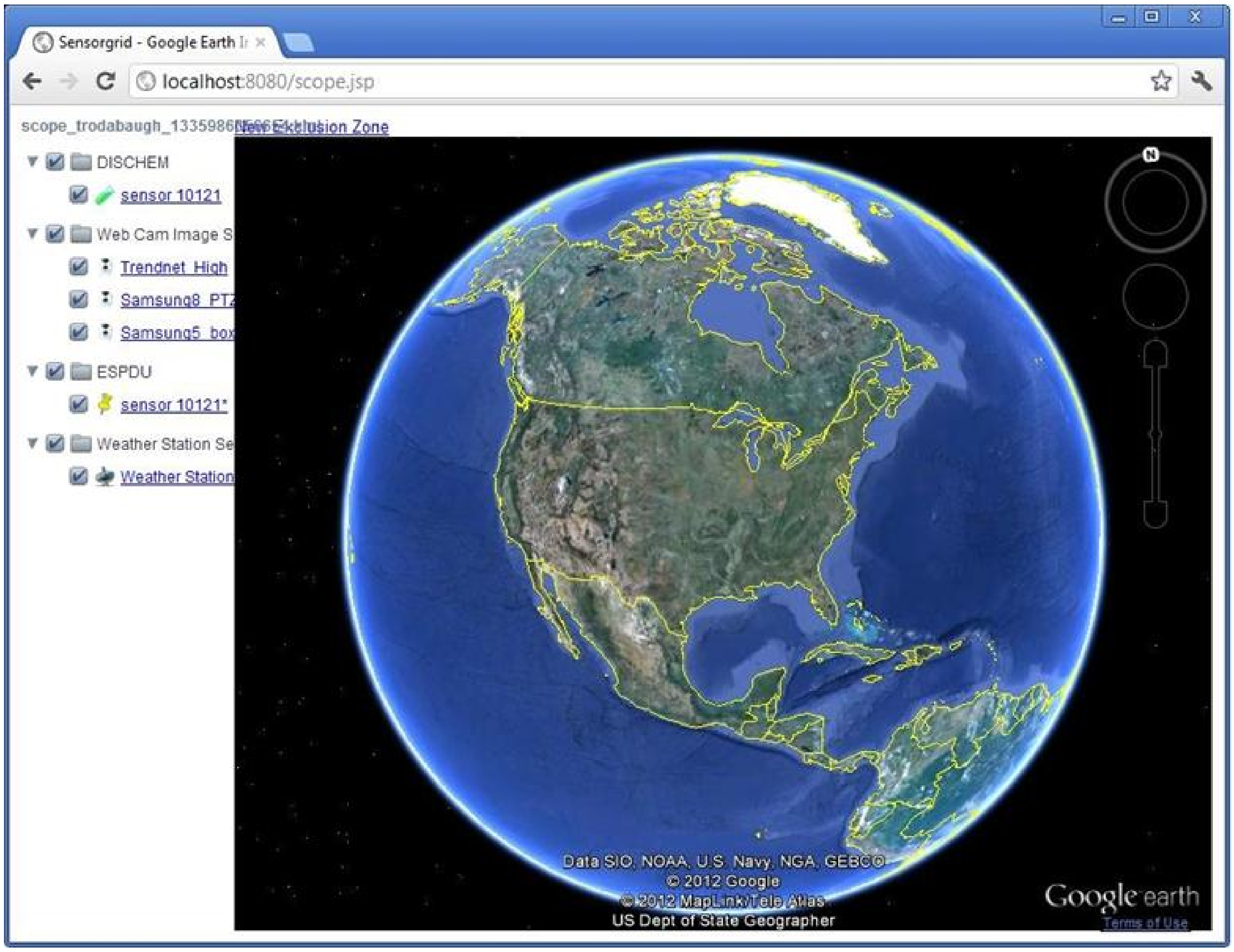


Figure 79. SCOPE View of the Google Earth Client

This client allows the user to visualize the GPS sensors in a geospatial display and provided the basis and prototype for the development of the SCOPE. The Google Earth client was only designed to subscribe and display GPS sensors from the Sensor Grid. This client also allows for a ten-step history tracking visualization, to allow the users to see historical data.

4.4.1.2.4 IP Camera Client

The purpose of the IP Camera client is to render the IP Camera sensor data that is being published to the Sensor Grid. The SCOPE IP Camera client interface is shown in **Figure 80**.

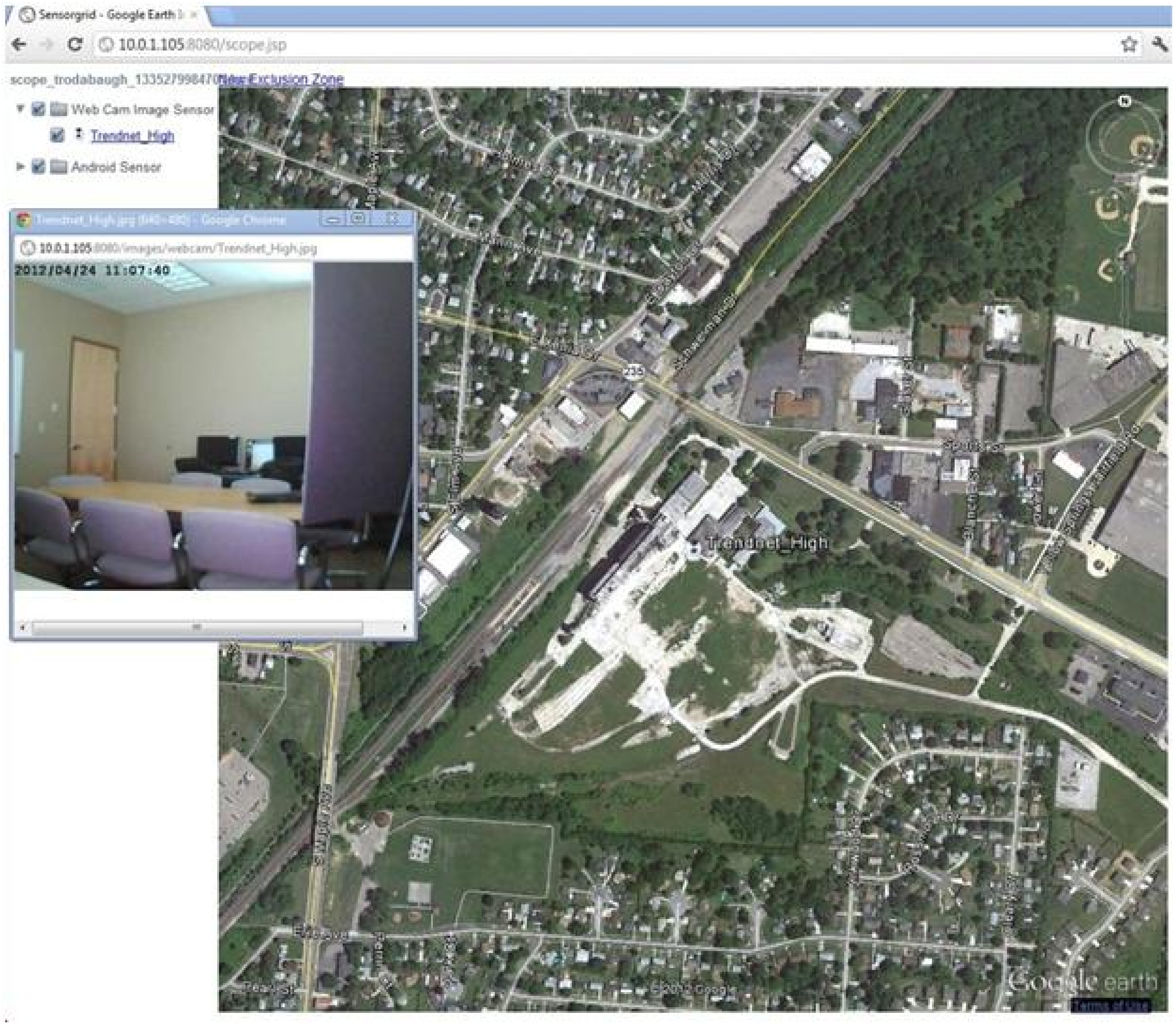

Sensorgrid - Google Earth
10.0.1.105:8080/scope.jsp
scope_trodabaugh_1335279984700 New Exclusion Zone
Web Cam Image Sensor
Trendnet_High
Android Sensor
Trendnet_High.jpg (640×480) - Google Chrome
10.0.1.105:8080/images/webcam/Trendnet_High.jpg
2012/04/24 11:07:40
Trendnet_High
Google earth

Figure 80. SCOPE View of an IP Camera Client

This client subscribes to all of the publishing IP Camera sensors, and includes a dropdown menu to select the different camera JPG streams. It also gives options to change the refresh rate of the IP Camera sensor, as well as send commands to the camera itself to move the direction of the camera if it is a pan/tilt/zoom (PTZ) type camera.

4.4.1.2.5 Mobile Platform Client

Early on in the research effort, the research team explored building a client that would enable the user to control a sensor. The Mobile Platform client is a software tool that allows the user to receive data from the platform and to enable them to send discrete commands to the Mobile

Platform. There are two versions of the client, one for the PC, and one for the Android tablets. The function of the two sensors is identical, however they differ in form.

4.4.1.3 Proxies

4.4.1.3.1 Android GPS Proxy

The purpose of the AndroidGPS proxy is to allow GPS data from Android devices to be published to the sensor grid. The AndroidGPS sensor is architecturally different from other sensor grid sensors. Since computing power and resources are limited on Android devices, Android devices must connect to the sensor grid at a much lower level via Narada Brokering. Thus, the AndroidGPS proxy serves not only a traditional sensor grid sensor but also as a proxy which elevates published data from the Narada Broker level to the sensor grid level. The raw bytes containing the GPS data are received by the proxy repackaged and published at the sensor grid level. A single instance of the proxy is capable of elevating data from multiple devices simultaneously. The proxy is aware of each Android device publishing GPS data. If the device publishes GPS data for the first time, a new sensor grid sensor is provisioned using the device name and the data is published. If an already provisioned device publishes data, then the existing sensor grid sensor is used. Stated another way, the proxy is capable of dynamically and intelligently provisioning sensor grid sensors whilst publishing their data.

4.4.2 Integration Phase

From the beginning of the Sensor Grid project, the focus was to take the Sensor Grid Middleware that the IU members of the team developed and integrate it together with the Ball team member’s developments, and create unique tools with special purposes. The first few developments from IU were mostly trivial sample programs that displayed the usefulness of the Sensor Grid in a contrite manner. These programs included a simple file transfer program, whose usefulness was actually underestimated, various bluetooth sensors, an earthquake sensor, and a few others. The file transfer program was interesting due to the fact that it used the Sensor Grid to multicast out a file to multiple clients.

Of this first batch of sensors, almost none of them were used in the final demo. The team decided to develop a set of sensors (identified by the LVC SIDFOT user) and integrate them into the Sensor Grid itself. The process of integration was difficult at first. The first sensor developed was the IP Webcam sensor, along with the IP Webcam client.

When developing a sensor or client for the Sensor Grid, it is often helpful to start with an example sensor, so that you can follow its design and implementation in the new model. When developing sensors or clients, it is also beneficial to build the opposite piece of the puzzle at the same time. That is to say, when developing an IP Webcam sensor, write the IP Webcam client at the same time. This allows the software to be tested at varying stages along the way without the need to rewrite a bunch of code if errors are found. This incremental development also enables testing to begin before the development is complete on both services to ensure that the sensor and client do indeed talk correctly together.

When developing a network of computers, thought needs to be paid towards security. Everything must be secure; the protocols, the ports, and the data must all be protected somehow. There was much discussion on security with IU. In the early stages, no thought was given to security. This was because it was a test network, in an ideal environment, where the only requirement was for it to work. When development on this network began, the team to realized that the test computers were being exposed to attacks by running this software. When the team began sitting down with IU researchers and talking about security, there was an initial backlash as they didn't agree that it was necessary. Once convinced of its importance, IU personnel implemented some security measures.

The first measure integrated into the Sensor Grid was encryption. There was no point in trying to lock anything down without secure computer transactions going on. This was implemented by adding a flag in the sensor properties that would specify the use of encrypted traffic. This encrypted traffic would connect between the sensors and the server and then between the server and the client. The data traffic is now fully SSL encrypted.

The second capability implemented to help lock down the Sensor Grid network was the incorporation of LDAP authentication into the Sensor Grid software. This was a little bit more work in terms of an integration process. The team needed to go back and rewrite several services and create an authentication mechanism that was not intended to exist in the original specification. Once this key piece was written, the application developers (Ball) went back through the sensors and clients that had already been written and updated them to enable LDAP authentication. Through forethought in the implementation approach, the IU developers set up the LDAP authentication in the Sensor Grid system so that only a difference in the command line string is required to start up the sensor in an LDAP authentication mode, or if it would be considered an "Anonymous" sensor or client. This allowed all of the LVC SIDFOT sensors and clients to be updated very rapidly and significantly reduced the need for excessive rework to capitalize on the new LDAP authentication capability.

4.4.3 Demonstration Phase

The Sensor Grid Test bed is a tool that is being used for the integration of Live, Virtual and Constructive sensors for the LVC-SIDFOT program. Inherently the Sensor Grid collects data from live sensors and shares it with those that have an interest and need for that data. LVC-SIDFOT uses the Cry Engine to simulate a scenario in an area that can overlap a real world location that is already covered by real sensors. Integrating the technologies developed for the Sensor Grid Testbed and the LVC SIDFOT program provides a research tool that can be used to investigate trust and trustworthiness algorithm development in conjunction with “First Responder” training research in a safe and repeatable fashion.

4.4.3.1 Mid-Review Demonstration

At the mid project review demo the focus was on a verifying form and validating trustworthiness of a sensor.

A scenario was set up where an object would travel through the scene in which there were system overlapping camera views along the path of the object. The object would come to rest behind an obstructing object. The camera pointing at the obstructing object could only see the entrance on the left and the exit from the obstructing object on the right. The point of the demo was to see if the operator could trust the implication that the object of interest was still behind the obstructing object.

To achieve a higher level of trustworthiness research, several tools in the sensor grid arsenal were added, including the use of IP webcams, PTZ IP webcams, the SACK tool and the mobile platform. During the live action of the demo, the team tracked the object of interest through the view path of the webcams that have a view of the path of the object. Then, given that the final location of the object can be estimaged, the PTZ webcam was moved to cover that location and confirm the object was there. For a second confirmation, the SACK tool was used to query the sensor’s historical data in the area of interest during the time frame of concern. The resulting data was assembled and sorted by time allowing the user to view the data. Additionally this allowed for replaying the data in normal speed, fast forward, stop, back up, and to scroll through manually. Using this tool, the user was able to track the object through the viewing areas of the various sensors to where it came to rest behind the obstructing object. Finally, the mobile platform, with IP webcam, was directed so it has a field of view of what is behind the obstructing object. This supported either confirming or rejecting the trustworthiness of the earlier data.

4.4.3.2 Final-Review Demonstration

SCOPE - Sensors

- Cry DIS – These are sensors in the virtual world of the Cry engine created for various scenarios. These sensors can be of the various sensors that are valid for the Sensor Grid test bed.
- Weather Station – Is a portable weather station from Columbia Weather Systems that is typically carried by first responders to disaster and hazardous situations. It will supply various weather data like temperature, relative humidity, dew point, heat index, wind chill, barometric pleasure and wind speed and direction. This data can be fed into plume calculation software to calculate an area of coverage of a potentially hazardous smoke cloud or chemical cloud.
- Android
    - GPS – An android tablet in the hands of a first responder can be used to track the location of that responder and verify their location is clear of a hazardous situation or see if they are close to a location that needs to be checked.
    - Picture – The Android device enables the first responder to send important picture data of the situation for evaluation that is Geo Located.
- Simulated Chemical Sensor – This is a simulated sensor that relates to the chemical issue of the scenario of interest.
- IP Web Cameras – Are devices that take image data of an area of interest and supplies it up for viewing via a TCP/IP based network. Several cameras were used in this research:
    - Four (4) Local cameras – At Ball's Dayton/Fairborn office, PT2cameras were used. Static cameras provide a set field of view and PTZ cameras have a much wider field of view because the direction of the camera can be changed through a much bigger field of view. They can also be mad to focus in an area and give more detail of the desired area.
    - Two (2) Calamity Ville Cameras – These cameras are mounted at fixed points on the side of the silos of the Calamityville facility.
    - Virtual Cameras – These are cameras that share images from the virtual world depicted in the Cry engine tool that match the real world. These cameras can take on some characteristics; they can look like static IP webcams or they can be

attached to moving objects in the scenario and provide images that the host objects would see as they move around the virtual world.

- SACK – This tool is a historical collection of all the data captured by the various sensors connected to the sensor grid test bed.

Vignettes

- SCOPE General Usage – In the first vignette, to the demonstrated capability shows the general abilities of the SCOPE tool. It shows how the sensors cover an area and provide situational awareness of the area encompassing the area if interest for the scenario. The tool shows what sensors are available, and how to jump to the area where sensors are located and to see details of the data being shared.
- SCOPE Timeout – This vignette demonstrated a feature of the SCOPE related to what happens when a sensor stops transmitting data or the grid stops receiving it. The vignette shows a sensor timing out, and how the resulting graphical depiction of the data shared by that sensor flag the data as out of date.
- Android Live GPS and Picture – This vignette demonstrated the correlation of live GPS data with the camera view of an Android device.
- SCOPE – Authenticated User and Un-Authenticated User– This vignette shows what an authenticated user can seed and do, versus what an un-authenticated user can't see or do.
- SACK – Observer Animator – This vignette demonstrated the ability to animate archived data retrieved from the SACK tool.
- Spoof Sensor – This vignette demonstrated a situation where a sensor can be spoofed thus allowing the researcher to run various trustworthiness experiments on a compromised sensor grid.
- Denial of Service Sensor – This vignette demonstrated the use of a denial of service sensor to support running trustworthiness experiments. For this vignette, the team created a tool to monitor the sensor grid. The demonstration used that tool to feed into a control element of the grid for managing such an attack. Next, the demonstration showed denial of service detection of the attack. This capability, like the Spoof Sensor, showed how these types of trustworthiness experiments can be imagined, realized and tested.

4.5 Final Sensor Cloud Performance and Results

4.5.1 Analysis of Single Message Broker Performance

The Sensor Cloud is supported by an underlying NB messaging system. NB is capable of being deployed in a distributed fashion (i.e., with multiple messaging nodes each carrying a fraction of the total system traffic). This section will discuss the results of analyzing the messaging performance of a single broker. Once the performance of a single message broker was understood, the team examined how the system performance scales by adding more brokers.

Using OpenStack, the team deployed the SensorCloud software to the FutureGrid cloud testbed (as illustrated in **Figure 81**). The team provisioned the Cloud controller with a single NB message broker and a single GB domain to host test sensors. Finally, multiple instances were launched to host a variable number of sensor clients.

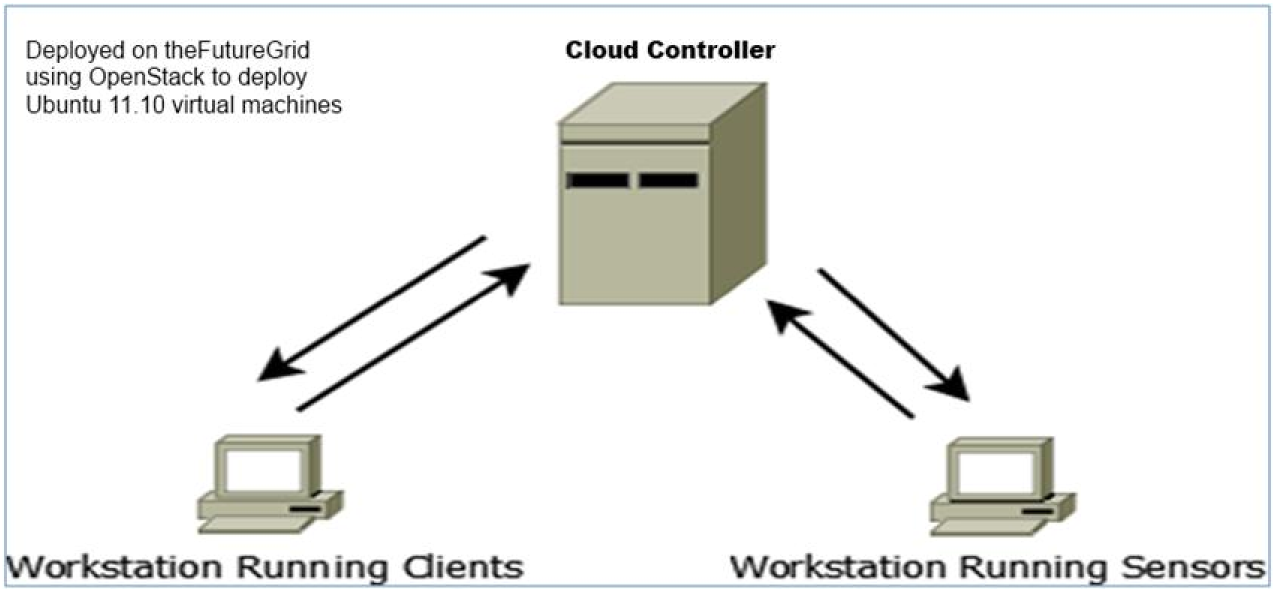

Deployed on theFutureGrid
using OpenStack to deploy
Ubuntu 11.10 virtual machines
Cloud Controller
Workstation Running Clients
Workstation Running Sensors

Figure 81. Futuregrid Set up for the Experiment.

The research team hosted the SGX 1.4 Sensor Grid middleware on the FutureGrid in four "large" instances.

- 2 cpu
- 6000MB ram
- 10GB disk

As a test case, the team simulated a video sensor publishing a typical real-time video stream. They selected the popular TRENDnet TV-IP422WN IP camera as the baseline. The TV-

IP422WN camera publishes audio/video data over an Real-time Streaming Protocol (RTSP) stream at a rate of approximately 1800Kbps when using the following encoding:

Video: codec MPEG4; width: 640; height: 480;
format: YUV420P; frame-rate: 30 frames/sec

Audio: codec PCM_MULAW; sample rate: 8000;
channels: 1; format: FMT_S16

In order to simulate video sensors of this type, the team published randomized data an average package size of 7680 bytes and an average publication rate of 30 packets per second (see **Figure 82, Figure 83** and **Figure 84**).

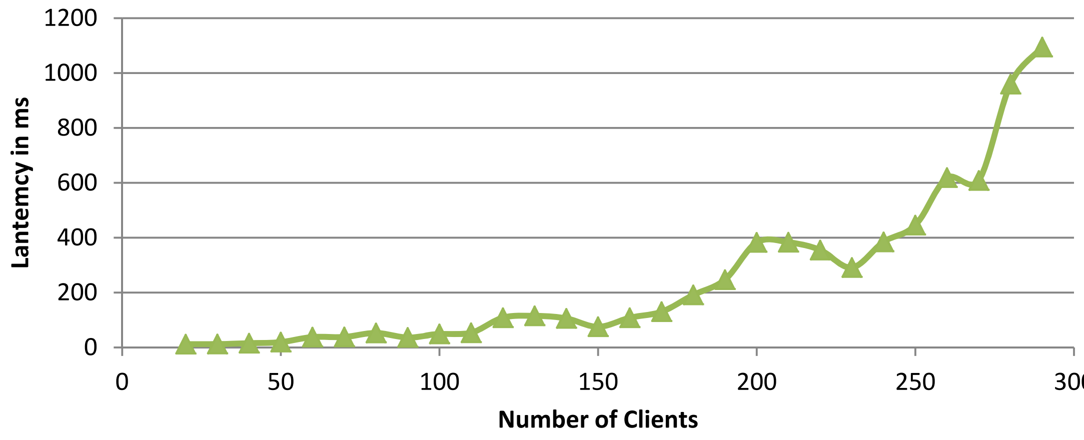


Figure 82. Performance Plot for a High-End Video Sensor with a Single Broker.

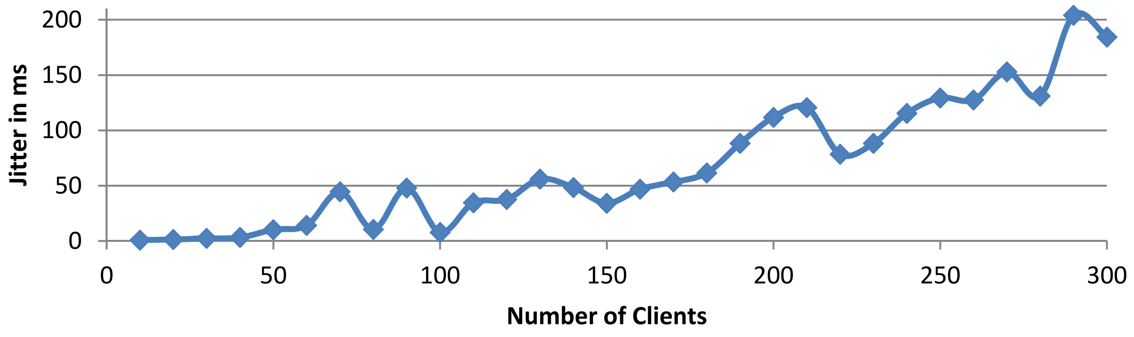


Figure 83. Average Jitter for a High-End Video Sensor with a Single Broker.

If only message delivery times are considered, one would conclude that a single broker is capable of supporting approximately 200 clients participating in a simulated video conferencing application. However, in real-time collaborative video applications, message latency is not the only factor; uniformity of the message latency must also be considered. In order to achieve a satisfactory user experience the video packets must also be delivered in a uniform (i.e., non-jittery) manner.

Considering this, the team also saw acceptable jitter until approximately 150 clients were reached. This figure is a better estimate of the true number of clients a single broker can effectively support. Finally, to further test this conclusion, the team also examined the jitter as a function of time (packet number).

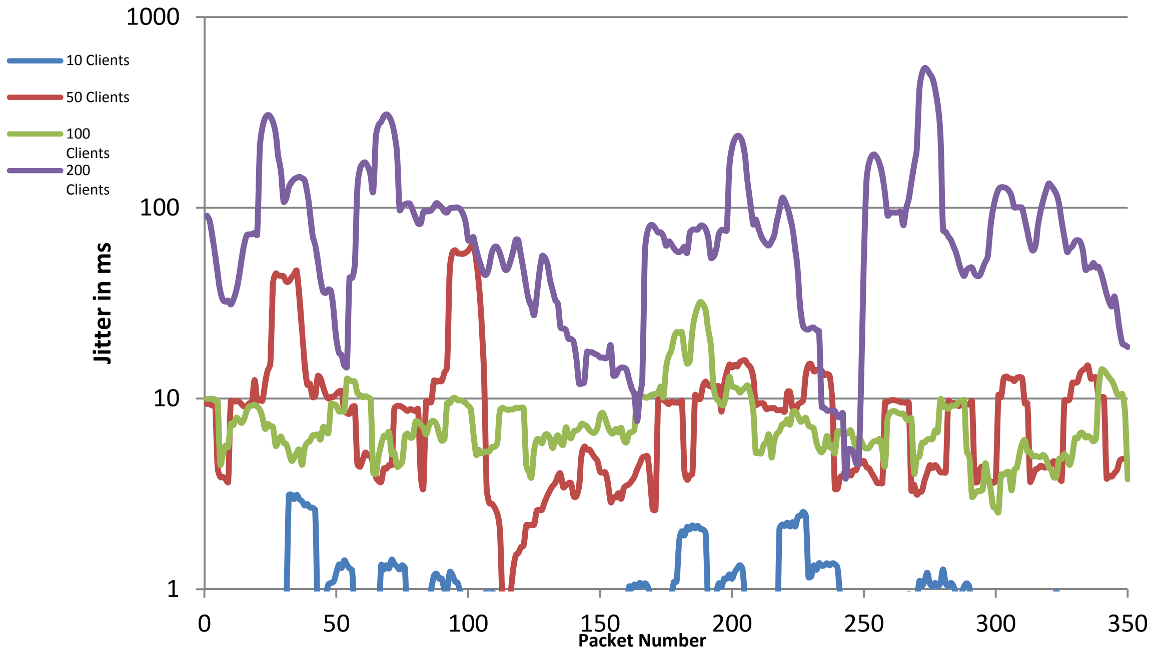


Figure 84. Jitter vs. Time for a High-End Video Sensor with a Single Broker.

As shown in Figure 84, the experiment demonstrated that a single broker is capable of supporting 150 clients participating in a real-time video conferencing application (where 640x480 video is streamed at 30 frames per second.)

Scalability Tests

Scaling can be achieved by deploying additional brokers to support larger client loads. Next, the team examined how Sensor Cloud performance scaling cases where multiple message brokers are used to load balance the messaging traffic. The results of these tests are shown in **Figure 85**. Refer to the NB Distribution User Guide for configuration details.

In the scalability tests, the team used the same high-end video sensor from the previous section, but also examined the case of GPS and standard video sensors. These three scenarios are:

- High-end Video Sensor: 30fps, 7680 byte packet (**Figure 86**)
- Video Sensor: 10fps, 1024 byte packet (**Figure 87**)
- GPS: 1fps, 1760 byte packet (**Figure 88**)

The results, summarized in Figure 86 - Figure 88, demonstrated how the Sensor Cloud can successfully scale to large sensor/client deployments by using a distributed broker scheme. For example, in the case of the High-end video sensors, one message broker can support ~200 clients, two message brokers will support ~400 clients, and five message brokers can handle ~1000 clients.

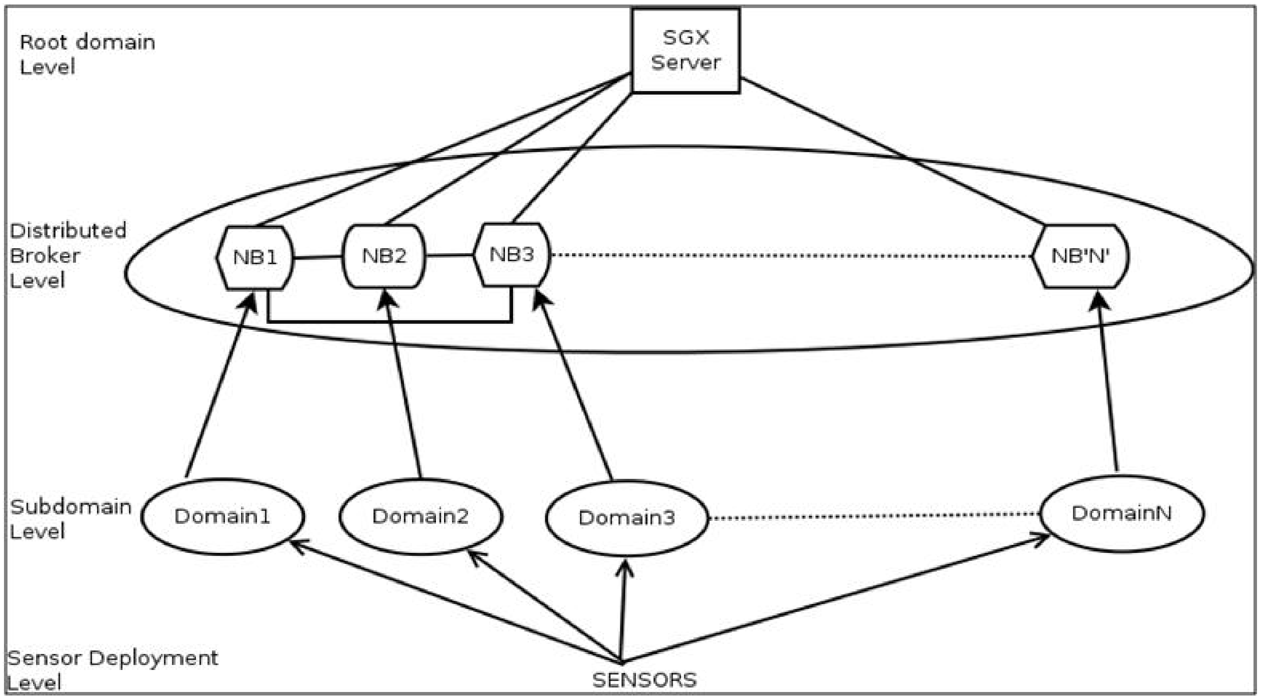


Figure 85. Multiple NB Brokers Load Balancing Sensor Messages.

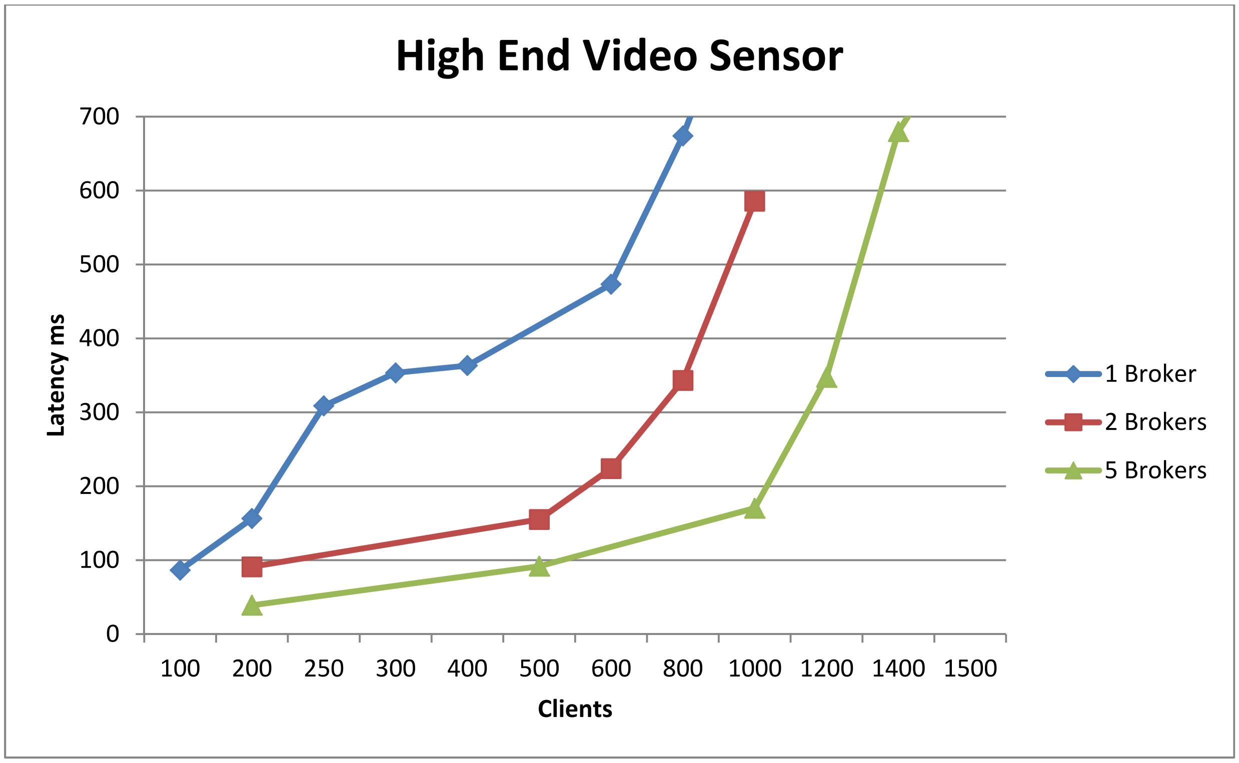


Figure 86. High-End Video Sensors with Multiple Brokers.

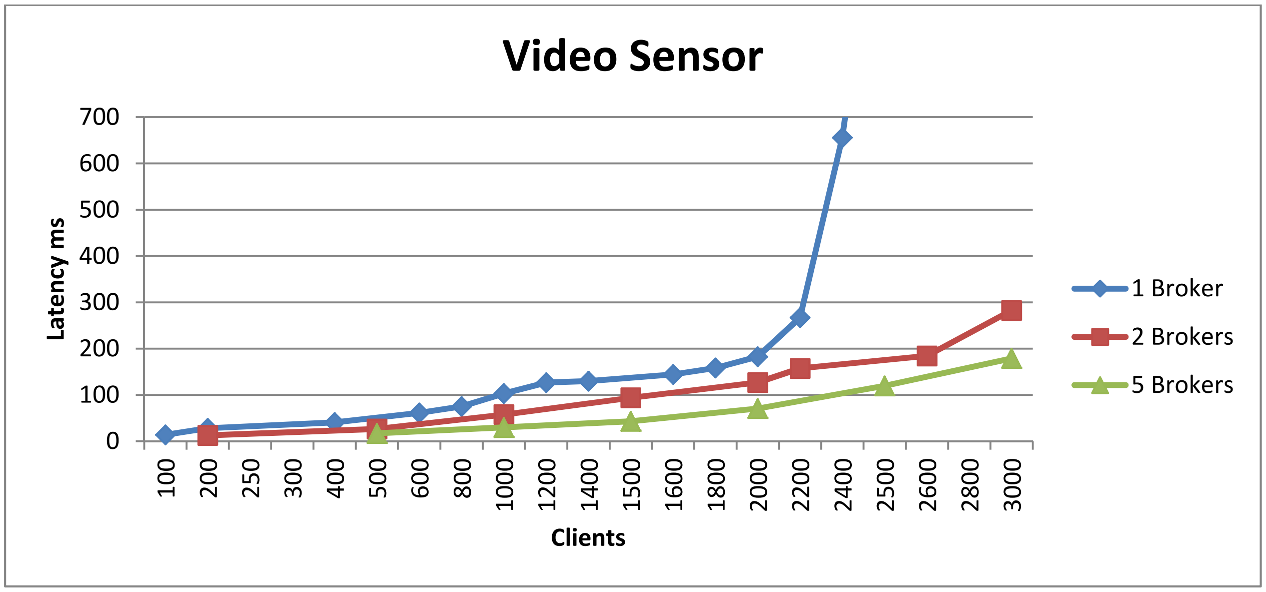


Figure 87. Standard Video Sensors with Multiple Brokers.

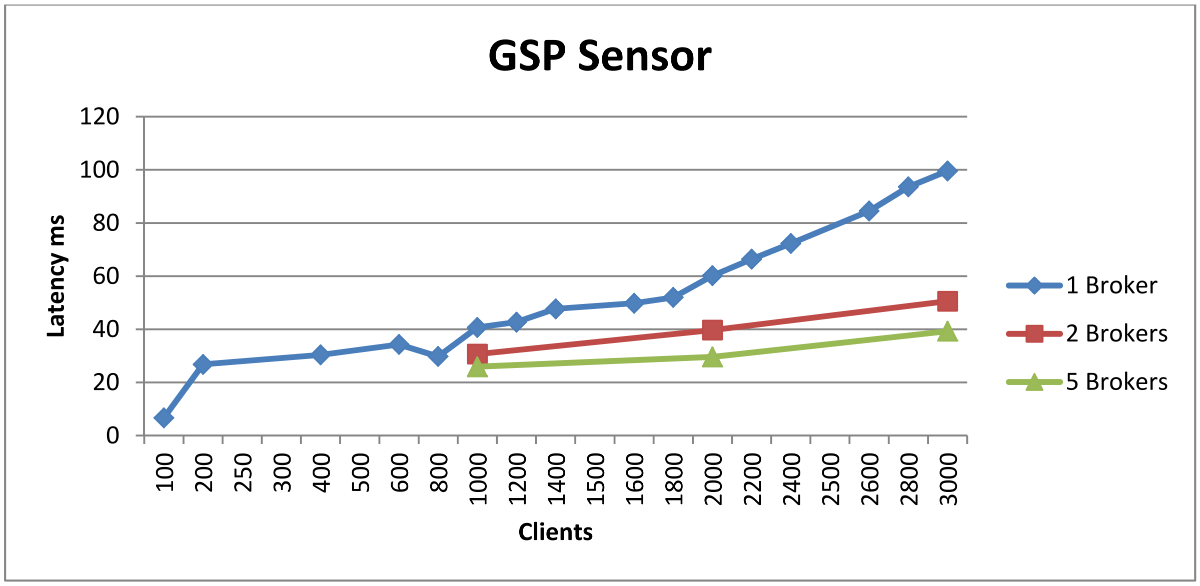


Figure 88. GPS Sensors with Multiple Brokers.

4.5.2 Middleware Analysis

This section discusses certain network factors which influence Sensor Cloud performance. With high-end video sensors, each sensor transmits ~2Mbps of data, therefore a message broker with a 100Mbps connection can, at best, only support 50 high-end video sensors. Correspondingly, a broker with a 1Gbps connection can theoretically support 500 high-end video sensors; however, in practice it can only support half that number.

Figure 89 compares the performance for a single message broker, running on the same physical hardware and virtual machine configuration. When sensor data saturates the underlying network this is the limiting factor in messaging performance. In cases where the network pipe is sufficiently large, system performance is a function of messages per second.

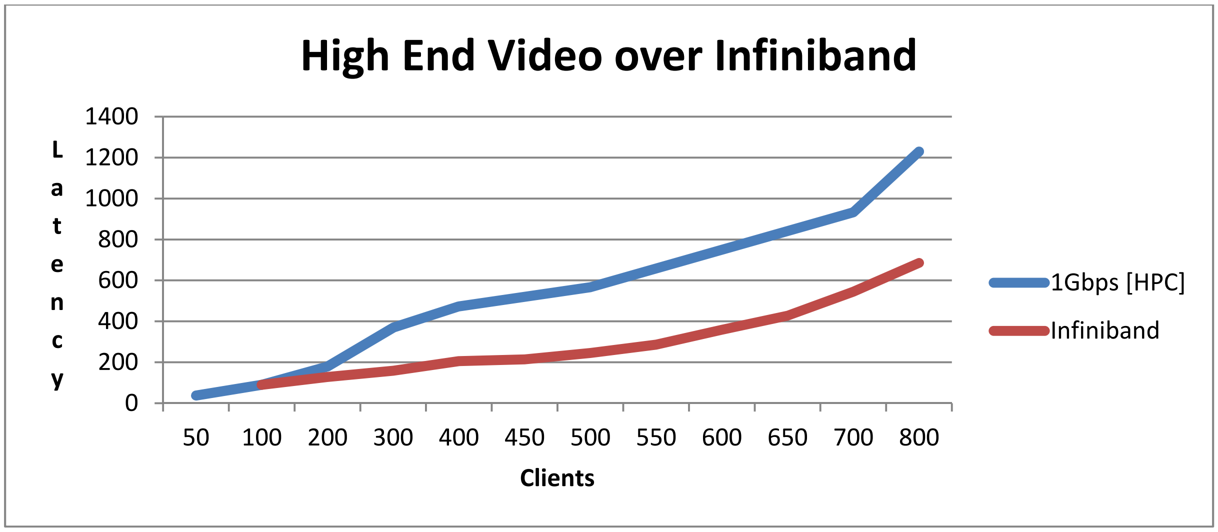


Figure 89. 1Gbps versus Infinand Over IP (10Gbps) Performance.

Another important networking consideration is the distance, and number of network hops between publishers and subscribers. These results were discussed earlier in section 4.1.8.5.

In **Figure 90** we determine the effect of distance and message latency. The distance between “India” and “Hotel” is 158 miles. The distance between “India” and “Sierra” is 1784 miles. The increase in Sensor Cloud message latency due to distance is seen to be just as predicted in earlier examination.

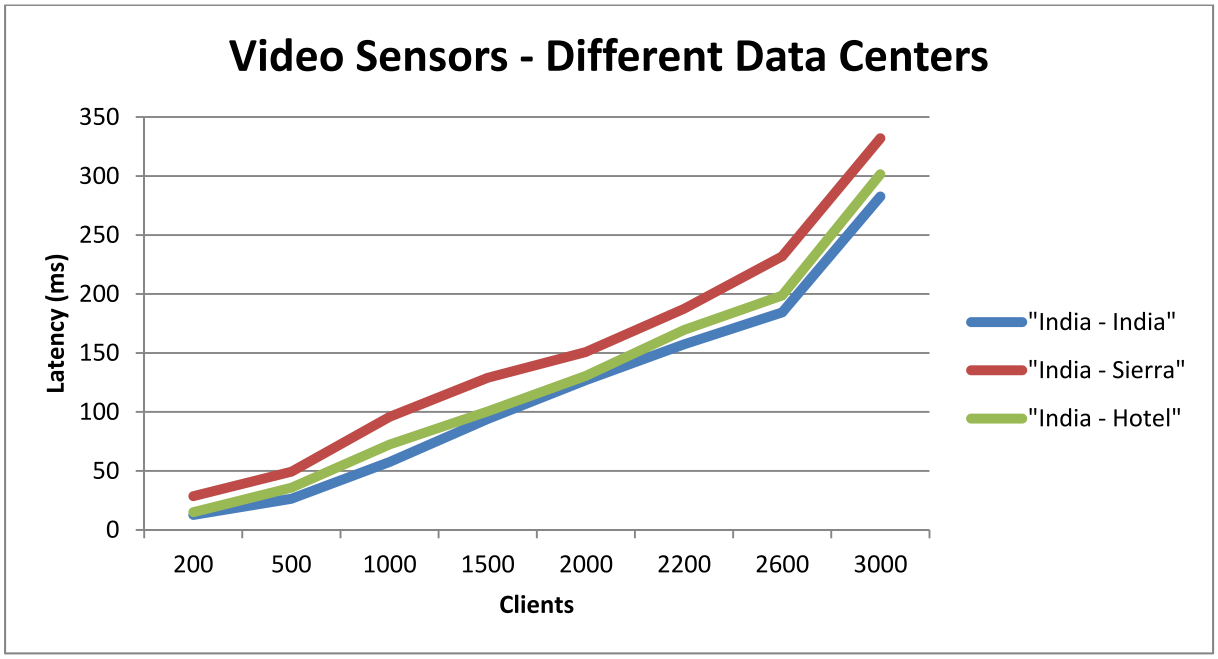


Figure 90. Geometric Effect on Message Latency.

5.0 Conclusion

5.1 Summary

This final report details the overall R&D activity accomplished on AMSA TO 4. The overall objective of the AMSA contract was to develop techniques, and an architecture, to help develop additional experiments aimed at ensuring trustworthiness and semantically correct interoperability among distributed sensor networks in support of multi-layered sensing. The primary focus of the TO 4 research, entitled “Advanced Technology Sensor Clouds,” was to conduct research, develop technology and components, and integrate the results for prototyping scalable cloud computing and advanced sensor management services into a Multi-Layered Sensor Grid testbed.

5.1.1 Sensor Cloud Research Summary

Early in the research effort, the research team conducted three types of experiments on FutureGrid, Amazon EC2 and an OpenStack-based private cloud and hybrid cloud to understand their respective network and performance characteristics in distributed clouds setting to support scalable collaborative sensor-centric applications. For these experiments, the team ported the Grid Builder to FutureGrid and developed virtual GPS sensors for managing the scaling of application-level deployed sensors to a large number. The team measured certain distributed clouds characteristics at the network, transport and application levels. The insight gained working through the complexity of the current state of cloud procedures, interfaces, platforms and virtualization technologies helped to lay a better foundation for the development phase of the work. The details of these early sensor cloud experiments are discussed in Section 4.1.8.5.

Later, toward the final segment of this research effort, additional experiments were completed to collect performance data for the final sensor cloud implementation. These results are discussed in Section 4.5.

5.2 Lessons Learned

5.2.1 Sensor Cloud Research Lessons Learned

Although this study is preliminary due to resource limitation, we observed satisfactory performance characteristics for network, CPU and memory demanding simulations that were used as research tools in the experiments. The coupling of a flexible sensor-centric grid framework with a heterogeneous distributed hybrid clouds infrastructure like a private cloud, a community cloud (e.g., FutureGrid), and Amazon or other commercial public clouds has the potential to effectively support the study of large-scale, collaborative sensor-centric applications that have stringent real-time and quality of service requirements.

Cloud technology and systems of various natures have gained popularity in the last twelve months, mostly for small non-mission critical departmental or enterprise applications looking for a cost-effective way of deployment. These applications generally need to deploy one instance only, or for a minority of organizations they may need to deploy a few independently running instances. Sensor grid applications have a much more stringent latency, performance, scalability, reliability and fault-tolerance requirement.

Latency

Cloud systems introduce and assemble a number of technologies that user applications rely on. For certain classes of sensor grid applications that include sensors with tight data streaming delivery requirements, it was not clear before this research what the major contributors to latency were.   The team was able to isolate cloud system level latency from network level latency in all three cloud types; organization-scale private cloud, national-scale FutureGrid, and Web-scale Amazon cloud. It was observed that cloud system software latency was sub-millisecond in unloaded cases and low milliseconds in loaded cases. Latency of sharing sensor streams was primarily due to the distance between sensor services and sensor applications. As a first order approximation, measured latency had a linear relationship with distance. From national-scale to Web-scale, the FutureGrid and Amazon EC2 exhibited attractively low network latency that could support some of the most demanding low-latency sensor grid applications. For instance, real-time sensor streams from GPS, remote robots, Webcams, or VoIP sensors are necessary for certain mission-critical deployment and tasks.

Bandwidth

National-scale clouds like the FutureGrid and Web-scale clouds like the Amazon EC2 offer on-demand bandwidth capacity that is better that 100 mbps LAN. Such bandwidth availability allows bandwidth-demanding sensor streams to be served effectively and on a timely manner. Coupled with the low latency observed, the current network characteristics of these clouds did not appear to be a potential bottleneck for larger scale sensors and sensor application deployments.

Scalability

Being able to scale up computing and network capabilities on-demand is an important requirement for large-scaled sensor grid applications. No organizations will want to over-reserve computing capabilities for estimated peak demand when most of the times the demand will be dynamic and sub-peak. The team's experiments did not attempt to push the limits of FutureGrid, but were meant for understanding the viability of scaling up computing resources on-demand. The results show that a national-scale cloud infrastructure such as FutureGrid could scale up from 1 Nimbus instance (equivalent to a 2-core Xeon X5570 with 12 GB RAM) to 111 instances (equivalent to 111 2-core Xeon X5570 with an aggregate 1.32 TB RAM) rapidly. This indicates that cloud technology and systems could be a natural fit for sensor grid applications, many of which are dynamic in nature.

Interoperability

In order to support Web-scale sensor grid application in the real world, the underlying sensor grid middleware and framework such as the Anabas SCGMMS must be able to support heterogeneity by design. In this study, the team assembled and worked with a heterogeneous experimental setting comprising of hybrid clouds and a plethora of cloud technologies. The results of the experiments with sensor grid applications and systems building that interoperability among heterogeneous and distributed components could be efficiently and effectively supported by abstracting every capability as a service, and communicating among services and applications via service message interfaces.

5.2.2 Sensor Grid Application Lessons Learned

Authentication

Authentication credentials for LDAP were specified in a central file. This approach was cumbersome and inflexible. A better approach and potential modification is to allow both clients and sensors to specify credentials dynamically at run time.

For sensors that use the sensor grid JAR file (the majority of sensors), any change to the authentication credentials means the sensor grid JAR files must be rebuilt and distributed to each sensor. Allowing sensors and clients to dynamically specify credentials at run time eliminates the need to redistribute the JAR files.

For clients that use LDAP authentication, the sensor grid, effectively, must be installed on each machine where the client executes. Allowing sensors and clients to dynamically specify credentials at run time eliminates the need to install the sensor grid on client machines.

Sensor and Client Provisioning

Both sensor and clients, when created, are created in a separate thread. Calling code should be aware that creation of sensors and clients is done asynchronously. Care should be taken when shared resources are accessed by the caller and the creator so that resource contention is avoided.

Sensors, when provisioned, read their connection information from a location labeled "%SENSORCLOUD_HOME%\config\mgmtSystemPrimary.conf" or possibly from the corresponding JAR file.

Development

Before beginning development, install and build the sensor grid so that the maven artifacts are stored on the local machine's repository. Then, one can use Netbeans and/or Eclipse to begin development. To develop a new project, it is recommended that the developer copy an existing project that is closest in similarity to the target project.

If one chooses to start a new Maven project, one should, whenever possible, add artifacts through Maven.

If one chooses to start a new standard Java project, then the traditional method of JAR files stored in a lib folder relative to the project should be used.

When sensible, it is a good idea to multi-thread sensor code so that, for example, if the sensor is busy publishing data, it can still respond to command and control requests.

Debugging and stopping at breakpoints for an extended period of time can cause a sensor's and/or client's connection to the grid to timeout. When a connection time out occurs, the sensor and/or client is typically shut down.

Doing 64-bit development requires a change be made to a sensor grid configuration file. 32-bit development with Java 1.6 has proven to be stable and is recommended.

Maven 3.0.x is recommended.

Ensure the development machine does not have multiple Java virtual machines installed.

The Grid Builder persists sensors in the Graphical User Interface even when they are de-provisioned. After a period of time, the stale sensor will read as UNREACHABLE in the GUI when selected. This anomaly in Grid Builder does not affect the messaging to client and sensors.

Grid Configuration

If one wishes to change the IP address to which sensors will connect, one can edit the file named "%SENSORCLOUD_HOME%\config\mgmtSystemPrimary.conf." Do a search and replace of the dated IP address with the new IP address. The developer then needs to run the startLocal.bat script from the machine on which the change was made before attempting to provision sensors as "startLocal.bat" copies the "mgmtSystemPrimary.conf" file.

LDAP connection information is hard-coded in the LDAPAuthentication.java file. Changes to LDAP information requires the alteration of this file and rebuilding of the grid. Distribution of the new JAR file to sensors and/or clients may also be necessary. When updating the host name or IP address in the LDAPAuthentication.java file, do not use 127.0.0.1 or localhost. Rather, one should use the actual host name or IP address of the machine so that if other machines attempt to read this information, it will remain valid.

Sensor Integration

When looking for sensors to integrate, it was difficult to find sensors that had the interface already integrated into the device package that could communicate with computers and the sensor grid. Usually it was the raw device that needed to be connected to a processor along with a way to communicate, ie Ethernet, WiFi, Blue Tooth, etc. This usually translated into the face that some type of hardware integration with the sensor was necessary before it could be added to the list of sensors integrated into the grid.

5.3 Recommendations

5.3.1 Sensor Cloud Recommendations

Future work for improvement includes a better understanding of how to fully utilize the potential of a single instance to confidently simulate the optimal or near-optimal number of sensors possible without worrying about system abnormality due risks of running out of resources in an instance. Scalability in terms of using more instances per cloud should be incorporated to augment scalability in the number of distributed clouds.

The underlying messaging system that was used for Grid of Grids and Sensor Grid to Sensor Cloud studies was the NaradaBrokering system developed by Indiana University. The NaradaBrokering system is well-designed and mature. It has been serving the research need very well. However, there are some other on-going open-source messaging systems that incorporate the latest technology and receive more supporting resources and feedback. It will be a worthwhile effort to try to substitute NaradaBrokering with more modern messaging system.

5.3.2 Sensor Grid Application Recommendations

Authentication

The team recommended adding an additional interface that would support an authentication approach which allows dynamic credential specification as outlined in the Lessons Learned –

Authentication section above. The interface would not change the current capability, but would add a new interface that could be used going forward so as to not break legacy code.

Grid Configuration

One aspect of the Grid configuration issue that leaves a legacy of a significant influence is the SYSTEM Environment variable. By using the environment variables you force a more complicated and involved installation on a computer or hand held device to access the sensor grid. As the implementation moves forward, the design needs to move toward interfaces that either don't require a significant installation process or no installation process at all.

Sensor Integration

Moving forward it would be helpful to select a small embedded platform that would allow sensors to be simulated on the device until actual sensors can be integrated with adequate hardware to allow real integration into the Sensor grid.

Create a standard software interface, to the sensor grid, that is based upon the hardware chosen to perform the sensor simulation previously discussed.

5.4 Conclusion

5.4.1 Sensor Cloud Conclusions

The initial results obtained in this research were encouraging in helping to lay a better foundation to build large scale, high-performance, low-latency, real-time collaborative sensor grid applications in clouds. The model of hundreds of millions of deployed sensors all over the world requires scalable, on-demand computing and communication capabilities in order to be able to harness into supporting the vision of multi-layer sensing to provide timely, actionable, trusted, and relevant situation awareness to decision makers at all levels of commands. The integration of distributed sensors and sensing systems operated and owned by different stakeholders will be best facilitated by exploring distributed and heterogeneous clouds.

6.0 References

1. Geoffrey Fox. *FutureGrid Platform FGPlatform: Rationale and Possible Directions (White Paper)*. 2010 [accessed 2010 June 12]; Available from: http://grids.ucs.indiana.edu/ptliupages/publications/FGPlatform.docx.
2. *FutureGrid Homepage*. [accessed 2011 January 19]; Available from: http://www.futuregrid.org.
3. Wenjun Wu, Geoffrey Fox, Hasan Bulut, Ahmet Uyar, and Tao Huang, *Special Issue on Voice over IP edited by John Fox, P. Gburzynski: Service Oriented Architecture for VoIP conferencing* Theory and Practice of the International Journal of Communication Systems April 13, 2006. **19**(4): p. 445-461. DOI:http://dx.doi.org/10.1002/dac.803. http://grids.ucs.indiana.edu/ptliupages/publications/soa-voip-05.doc
4. Shrideep Pallickara, Hasan Bulut, Pete Burnap, Geoffrey Fox, Ahmet Uyar, and David Walker. *Support for High Performance Real-time Collaboration within the NaradaBrokering Substrate*. 2005 May [accessed 2011 March 11]; Available from: http://grids.ucs.indiana.edu/ptliupages/publications/NB-Collaboration_update.pdf.
5. Ahmet Uyar and Geoffrey Fox, Investigating the Performance of Audio/Video Service Architecture I: Single Broker, in IEEE International Symposium on Collaborative Technologies and Systems CTS05. May, 2005, IEEE. St. Louis Missouri, USA. pages. 120-127. http://grids.ucs.indiana.edu/ptliupages/publications/SingleBroker-cts05-submitted.PDF. DOI: http://doi.ieeecomputersociety.org/10.1109/ISCST.2005.1553303.
6. Ahmet Uyar and Geoffrey Fox, Investigating the Performance of Audio/Video Service Architecture II: Broker Network, in International Symposium on Collaborative Technologies and Systems CTS05. May, 2005, IEEE. St. Louis Missouri, USA. pages. 128-135. http://grids.ucs.indiana.edu/ptliupages/publications/BrokerNetwork-cts05-final.PDF. DOI: http://doi.ieeecomputersociety.org/10.1109/ISCST.2005.1553304.
7. NaradaBrokering. *Scalable Publish Subscribe System*. 2010 [accessed 2010 May]; Available from: http://www.naradabrokering.org/.
8. 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.
9. 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.
10. 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.
11. Fox, G., *Grids of Grids of Simple Services.* Computing in Science and Engg., 2004. **6**(4): p. 84-87. DOI:10.1109/mcse.2004.10. http://grids.ucs.indiana.edu/ptliupages/publications/Cisegridofgrids.pdf
12. Geoffrey Fox. *Cloud Computing for ADMI*. 2010 [accessed 2011 March 11]; ADMI Board Meeting and faculty workshop at Elizabeth City State University Available from: http://grids.ucs.indiana.edu/ptliupages/presentations/ECSU-Dec16-10.pptx.
13. *TeraGrid open scientific discovery computational infrastructure*. [accessed 2010 November 20]; Available from: https://www.teragrid.org/.
14. Geoffrey Fox. *Interview on FutureGrid*. 2009 September 29 [accessed 2011 March 11]; by Sander Olson Available from: http://nextbigfuture.com/2009/09/interview-of-geoffrey-fox-director-of.html.
15. Nurmi D., Wolski R., Grzegorczyk C., Obertelli G., Soman S., Youseff L., and Zagorodnov D., The Eucalyptus Open-Source Cloud-Computing System, in 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. CCGRID ‘09. 18-21 May, 2009. Shanghai. pages. 124-131. DOI: 10.1109/CCGRID.2009.93.
16. *Eucalyptus Open Source Cloud Software*. Available from: http://open.eucalyptus.com/.
17. *Nimbus Cloud Computing for Science*. [accessed 2011 March 11]; Available from: http://www.nimbusproject.org/.

18. Ping computer network administration utility used to test the reachability of a host on an Internet Protocol (IP) network and to measure message round-trip time. [accessed 2011 March 20]; Wikipedia Entry Available from: http://en.wikipedia.org/wiki/Ping.
19. Tim Szigeti and Christina Hattingh, *Quality of Service Design Overview*. 2004: Cisco Press. http://www.ciscopress.com/articles/article.asp?p=357102&seqNum=3
20. Harshawardhau Gadgil, Geoffrey Fox, Marlon Pierce, Shrideep Pallickara. HP Search: Service Management & Administration Tod Abstract for VLAB Meeting, Minnesota, July 21-23, 2005.
21. WS-Context from OASIS http://www.oasis-open.org/committees/download.php/9904/ws-context.zip, November 2004.

LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

Acronym	Description
AMSA	Adaptive Multi-Layered Sensing Architectures
AFRL	Air Force Research Laboratory
AFRL/RY	Sensors Directorate
AFRL/RYW	Integrated Electronic & Net-centric Warfare Division
AFRL/RYWB	Trusted Avionics Systems Network Branch
AFRL/RYWC	Distributed Collaborative Sensor Systems Technology Branch
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
AWS	Amazon Web Service
BATC	Ball Aerospace & Technologies Corp.
CONOPS	Concept of Operations
COP	Common Operational Picture
COTS	Commercial-off-the-Shelf
CPU	Central Processing Unit
DIS	Distributed Interactive Simulation
DMOT	Distributed Mission Operations Training
DNS	Domain Name System
DoD	Department of Defense
DTIC	Defense Technical Information Center
EAR	Export Administration Regulation
EC2	Elastic Compute Cloud
Eucalyptus	Elastic Utility Computing Architecture Linking Your Programs To Useful Systems
GB	Grid Builder
GIG	Global Information Grid
GIS	Geographic Information System
GOTS	Government-off-the-Shelf
GPS	Global Positioning System
GUI	Graphical User Interface
HSN	Heterogenous Sensor Network
HPC	High Performance Computing
HTML	Hypertext Mark-up Language
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ISR	Intelligence Surveillance and Reconnaissance
ITAR	International Traffic in Arms Regulation
IU	Indiana University

Acronym	**Description**
JMS	Java Messaging Service
JPEG or JPG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KML	Keyhole Markup Language
LDAP	Lightweight Directory Access Protocol
LRF	Laser Range Finder
LVC	Live, Virtual, and Constructive
M&S	Modeling & Simulation
MOM	Message-Oriented Middleware
NB	Narada Broker
NCES	Net-Centric Enterprise Services
NTP	Network Time Protocol
P2P	Peer-to-peer
Pub/Sub	Publish/Subscribe
RFID	Radio Frequency Identification
R&D	Research & Development
RSA	Registered Service Adapter
RSS	Really Simple Syndication
RTT	Round-Trip Time
SA	Service Adapter
SACK	SensorGrid Archive Collector
SCGMMS	Sensor-Centric Collaboration Grid Middleware Management System
SCMS	Sensor Container Management Services
SCMW	Sensor Cloud Middleware
SCOPE	ScensorGrid Client for Operational Awareness
SCP	Sensor Client Program
SG	Sensor Grid Server
SHC	System Health Check
SIDFOT	Sensors Integration for Data Fusion in Operations and Training
SSA	Sensor Service Adapter
SSAL	Sensor Service Abstraction Layer
SSH	Secure Shell
SOW	Statement of Work
TCP	Transmission Control Protocol
TO	Task Order
VM	Virtual Machine
VoIP	Voice over Internet Protocol
UDOP	User-Defined Operational Picture
UDP	User Datagram Protocol
UL	Universal Locator

Acronym	**Description**
UML	Unified Modeling Language
UUID	Universally Unique Identifier
WBS	Work Breakdown Structure
WiiMote	Wii Remote Controller
WPAFB	Wright-Patterson Air Force Base
WS	Web Service
XML	Extensible Markup Language

Appendix A

Secure Cloud Computing with Brokered Trusted Sensor Networks

Apu Kapadia, Steven Myers, XiaoFeng Wang and Geoffrey Fox
School of Informatics and Computing
Indiana University, Bloomington
{kapadia, samyers, xw7, gcf}@indiana.edu

ABSTRACT

We propose a model for large-scale smartphone based sen- sor networks, with sensor information processed by clouds and grids, with a mediation layer for processing, filtering and other mashups done via a brokering network. Final aggregate results are assumed to be sent to users through traditional cloud interfaces such as browsers. We conjecture that such a network configuration will have significant sensing applications, and perform some preliminary work in both defining the system, and considering threats to the system as a whole from different perspectives. We then discuss our current, initial approaches to solving three portions of the overall security architecture: i) Risk Analysis relating to the possession and environment of the smart- phone sensors, ii) New malware threats and defenses in- stalled on the sensor network proper, and iii) An analysis of covert channels being used to circumvent encryption in the user/cloud interface.

KEYWORDS: Sensor Network, Brokered Network, Se- curity, Wireless.

1. INTRODUCTION

We consider systems in which there are large groupings of sensors reporting exorbitant quantities of potentially sensi- tive data, and the need to perform large amounts of process- ing or computation on this data with multiple large grid and cloud computing installations. The processing may need to be done in real or near-real time. Further, we consider that there are adversaries that have a vested interest in ei- ther learning information from the system, modifying the results finally output from the system (be it through modi- fication of the sensor input, filtering or processing of data), or denying access to the system. Therefore maintaining data provenance, secrecy and trust is of paramount importance throughout the data life-cycle (i.e, from the point of data collection by the sensors, to its final consumption by an individual or process). All data-transformation and filtering, networking and sensor aspects of these systems are assumed to be susceptible to attack. Similarly the environment in which some parts of the system operate is assumed to be potentially under adversarial control. In our modeling we assume the actual cloud-computing facility to be secure. Our goal is to be able to provide reliable results computed from sensor data in a manner that enables one (be it the user or the system) to make educated decisions on the reliability of that data based on trust metrics, while simultaneously preventing the loss of data-secrecy or integrity. Further, maintenance of system integrity and security is considered a core requirement. Issues such as anonymity are beyond the scope of our current research. Herein we provide a for- mal description of the networking architecture we antici- pate and the security threats. We delineate between threats and security holes for which conventional security technology suffices to solve the problem, those threats for which modifications to conventional technology are required, and those which are new and somewhat specific to the problem at hand. We next outline a largescale feasible research pro- gram to solve the many associated problems. We conclude by highlighting several of the aspects of this program for which we are actively engaged in producing solutions, and the architectures for our solutions.

1.1. Roadmap

In Section 2. we provide a high-level specification of the type of systems we are considering. This is followed, in Section 3., by a high-level threat model that depicts ways adversaries can manipulate such systems and their mal- leable environments. In Section 4., we provide more in depth discussions on three specific subsets of security prob- lems from Section 3. for which we are currently developing solutions. In Section 5. we provide related work for these problems. Section 6. finishes off with discussion and conclusions.

2. COMPUTATION, NETWORKING & SENSING MODEL

We consider a model in which there are potentially millions of deployed sensors. The sensors may be (but are not necessarily) organized by some principle into different hierarchical layers or partitions. These sensors may be continuously publishing their observations, or supply their observations on request. In either event the observations are relayed through a brokering and filtering network, where sensor data is eventually consumed by a cloud or grid-computing infrastructure; alternatively the data can be fil- tered or processed, and stored. Importantly, we do **not** consider traditional low-power sensors such as motes, RFIDs and smartdust, where a great preponderance of wireless sensor-network research has been done. Rather, we con- sider potentially high throughput sensors attached to a — in comparison — large amount of computational and networking power, e.g., in the cloud. Specifically, we consider smartphone-class devices with reliable cellular network connectivity (with hundreds of Kbps throughput as opposed to tens of Kbps available on motes) and frequent recharging (e.g., nightly) that supports more computation- ally intensive applications than motes. Yet, this model still leaves open a large number of security issues that must be solved.

In Fig. 2. we visualize the different components of the networked system. Android smartphones denote the sen- sors in the system, and are in the possession of individu- als. The smartphones have some computational capacity, and transmit through WiFi or cellular services to a broker- ing network, running over traditional TCP/IP services. The brokering service can itself have computers performing fil- tering, processing and/or creating other mashups of sensor data.

2.1. The Sensor

Herein, we consider the sensors to be modern smartphones. These devices are diversely deployed in the field, con- tain a large number of sensors, and have moderate com- putational ability. Further, they are fully networked, and with modern 3G networks have reasonable bandwidth (e.g., 100–1000kbps). Additionally, most sensors have 802.11 WiFi radios, and may have sporadic or continuous WiFi connections in urban environments, with bandwidth of 1-50Mbps. These phones may be in the control of trusted (or semi-trusted) individuals, or be located in some poten- tially untrusted environment. Further, they have a reason- able processing capability on modern low-power proces-

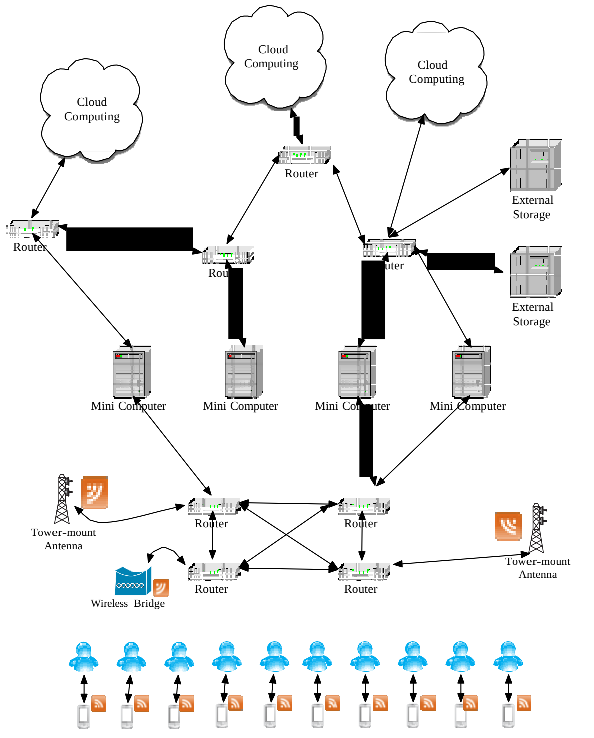


Figure 106. A Depiction of the Different Components of the Sensor and Cloud-Computing Network.

sors, such as an ARM architecture processor running at 500–800MHZ. It is assumed that the phones have standard sensors including, eGPS, 802.11x, Bluetooth v2 (Class 1, 2 or 3), temperature, orientation, acceleration, audio microphone, and camera (stills or video). In particular, our project focuses on the use of HTC G1 Android (v1.6) development phones, due to the ease of programming and their ability to multi-task (unlike the iPhone). Such plat- forms can perform a full host of cryptographic operations, but also have security issues relating to the fact that they are multi-purpose computing platforms. Thus OS secu- rity issues are larger, and it is difficult to construct a small OS, such as TinyOS [19] designed for motes, which can be more easily hardened to withstand attack. While the smart- phones are capable of more standard cryptographic proto- cols, a large number of such sensors in a region that are broadcast could overwhelm communications channels, and battery life is still a concern — if not as pressing. There- fore, low bandwidth and energy usage requirements are still a concern. However, one can easily port low-energy and bandwidth secure networking stacks, such as those pro- vided by TinySec [17] or MiniSec [21].

2.2. The Brokering Network

With potentially millions of smartphone sensors producing data at any given time, the need for a high performance networking infrastructure that is capable of self-filtering unimportant data feeds before they are transmitted for pro- cessing becomes apparent. Further, the need to funnel po- tentially very large amounts of bandwidth to a few collec- tion points for processing is also evident. The communi- cation between the sensors and the computing infrastruc- ture is mediated by a brokering network that uses a pub- lish/subscribe model. In such a model, each sensor can publish the data it is collecting on a continuous basis, along with appropriate meta-data that depict the content, prove- nance and trustworthiness of the data. Requests for specific information at the cloud or grid computing interface will drive the request for specific types and trustworthinesses of data from the sensors. Such requests will further invoke the subscription to different forms of data both real-time and stored. Typical forms of data cleaning and processing can, of course, be performed by dedicated servers who indepen- dently subscribe to sensor feeds, and then publish their own mashed data feeds for consumption by others. In such cases provenance and trustworthiness must be maintained. Ul- timately, there will be many different parallel consumers of data, and thus the network must be as responsible as is possible to prevent duplication of effort, redundant routing, streaming and processing of data.

For this project, the Narada Brokering network[1] is being used. The network can provide basic secrecy and integrity requirements, but does not by default provide any informa- tion regarding provenance or trustworthiness. While other suitable brokering networks can be used (e.g., Solar [?]) we chose Narada because of local expertise and support avail- able to our project.

2.3. Computing Model

We assume that the final consumers of data will be cloud or grid computations, as will many of the filtering and pro- cessing modules. While each cloud or grid may see its out- put as the final consumable, the desire to recycle computa- tion means that the data may itself become simply another input to an alternate computation upstream. The study of securing cloud and grid computation are separate research fields in their own right, and so our model simply assumes that these computations do not leak information, break in- tegrity of the data nor provide covert channels to the data. Computational power and storage is considered to be more or less limitless to within reasonable bounds.

[1]See www.Naradabrokering.org

3. SECURITY, PRIVACY & TRUST IS- SUES

The computing environments of a sensor grid are fraught with different kinds of threats, which endanger the security and privacy assurance the system can provide. Mitigation of these threats relies on establishing trust on individual system layers through proper security control. In this sec- tion, we survey the security and privacy risks on each layer of senor-grid computing and the technical challenges for controlling them.

A sensor grid interacts with its operating environment through a set of sensors. Those sensors work either au- tonomously or collaboratively to gather data and dispatch them to the grid. Within the grid, a brokering system fil- ters and routes the data to their subscribers, the clients of the sensor grid. We now describe the security and privacy issues on each layer of such an operation. This includes the environment the sensors are working in; the sensors; the grid; the clients; and the communications between the sensor and grid, and the grid and clients.

The Environment. An adversary could compromise the sensors' working environments to contaminate the data they collect. For example, one can add ice around individ- ual sensors to manipulate the temperatures they measure; alternatively, one could imagine that GPS signals were be- ing spoofed in an area. Detection of such a compromise can be hard, when the adversary has full control of the en- vironment. A possible approach is to check the consistency of the data collected from multiple sensors and identify anomalous environmental changes as indicated by the data.

Sensors. Sensors can be tampered with by the adversary who can steal or modify the data they collect. Mitigation of this threat needs the techniques that detect improper opera- tions on the sensors and protect its sensitive data. Since we assume sensors are smartphones, they also are susceptible to a large number of security concerns of traditional PCs, which includes viruses and malware.

Cloud or Grid. Information flows within the grid can be intercepted and eavesdropped on by malicious code that is injected into the system through its vulnerabilities. Authen- tication and information-flow control need to be built into the brokering system to defend against such a threat.

Client. The adversary can also manage to evade the secu- rity and privacy protection of the system through exploit- ing the weaknesses of the clients' browsers. The current design of browsers is well known to be insufficient for fending off attacks such as cross-site scripting (XSS) and cross-site request forgery (XSRF). Such weaknesses can be used by the adversary to acquire an end user's privileges to wreak havoc on the grid. Defense against the threat relies on design and enforcement of a new security policy model that improves on the limitations of the same origin policy adopted in all of the mainstream browsers.

Communication Channels. The communications be- tween the sensors and the brokering network, the broker- ing network and the cloud or grid, and the cloud or grid and the client, are subject to both passive (e.g., eavesdrop- ping) and active (e.g., man-in-the-middle) attacks. Coun- tering this threat depends on proper cryptographic proto- cols that achieve both data secrecy and integrity. In each case, different engineering requirements based on differ- ing scarce resources require different solutions. In the case of the wireless connection between the sensor network and the brokering network, bandwidth and power-usage are key requirements. Once on the brokering network, data prove- nance becomes a key challenge. Traditional cryptographic protocols would seemingly suffice from the cloud to the user. However, a tricky issue here is the information leaks through side channels. For example, packet sizes and se- quences. Our preliminary research shows that such infor- mation reveals the state of web applications, which can be further utilized to infer sensitive data within the applica- tion. Understanding and mitigating the problem needs fur- ther investigation.

4. PROBLEMS TO BE ADDRESSED

While there are a large number of potential security issues to be addressed, as partially scoped and enumerated in the previous section, the investigators are working on the fol- lowing specific problems.

4.1. Detection of anomalous use of sensors

A key issue involved in trusting data from the sensors in the described network is to ensure that the sensors themselves can be trusted. That is, either they are in the possession of individuals who are trustworthy, or they have not been tampered with in their environment if not possessed by an individual.

In our model if the sensor is in the possession of a trusted individual, it is more likely that its sensors are reporting an honest or legitimate environment, and not one that has been manipulated with the goal of producing faulty results that get incorporated in to final computation. Smartphones, however, can be easily stolen, misplaced or temporarily in- tercepted and reprogrammed by adversaries. If stolen or misplaced, the environment that the sensors report may be altered, and thus the data collected may be untrustworthy. The use of traditional authentication technologies to ensure a legitimate user is in control of the smartphone sensor is not practical, as said users cannot be queried to authenticate every time the sensor-net needs to report readings.

We propose a system in which a phone attempts to deter- mine if it is or is not in the possession of a legitimate user. In cases where the phone determines it is in questionable hands it deauthenticates itself. Deauthentication either re- moves it from the sensor network, or forces its sensor read- ings to be tagged as untrustworthy, with risk measurements being included in provenance data to ensure that the risk of improper readings is communicated down stream and taken into account on further processing. In order for the phone to determine whether it is under legitimate possession, we are developing a risk assessment system based on the inputs from the sensors of the phone itself. Thus the sensors are used directly to determine if the sensors' readings should be trusted. We are implementing a prototype of this system on the HTC/Google G1 Android (v1.6) Phone.

We are taking different approaches with different sensors on the phones. Note we are using these sensors to de- termine risk of improper possession independent of which sensors are of interest to the sensor network. Further, we make two broad classifications of the use of sensor input for risk determination. First, *environmental sensors* attempt to measure properties of the environment around the phone, or of the user. Second, *social-networking sensors* measure "friendly" or "unfriendly" people that surround the phone.

4.1.1. Environmental Sensors

Positioning Information. Android smartphones can de- termine their position using a combination of several differ- ent information sources, which includes cellular transmis- sions (in particular, tower location), GPS positioning and WiFi positioning. The combination of all of these pieces of information is often called eGPS, and frequently provides position far more accurately than any of the technologies alone. Our high-level goal is for the phone to learn certain geographic locations and routines that correspond to either a safe or dangerous state.

We extend the work of Farrahi and Gatica-Perez [14]. We are using a third-order Hidden Markov Model (HMM) to determine the risk of misuse of a phone based on current positional information. Farrahi and Gatica-Perez consid- ered the problem of determining location for contextual ap- plication purposes, but without specific interest in authen- tication and security mechanisms. A day is divided into blocks of 30 minutes. In any given period the phone is considered to be in one of four specified places (e.g., Home, Work, Aux 1, No Location Reading) or in a generic unlabeled place (Other). Thus the location of an individual through a time period is being converted into a string, as is depicted in Fig. 2. Currently, we are considering a super- vised learning case where a user specifically defines these five locations, with the goal of using clustering algorithms to eventually learn popular locations. Traces of individ- uals' positions are then collected, and the HMM iterative Viterbi training and Forward algorithm are used for train- ing on this past annotated data sequences and predicting risk. Based on a trained HMM, and a recent history of the phones' positions, the forward algorithm is used to deter- mine the likelihood of the recent history, and this estimate is used to determine the risk associated with the phone's current position. Of clear importance is the efficiency with which both training and evaluation can be performed. Due to the need to only occasionally perform training (say daily or weekly to update the movement model with the most re- cent trends), its efficiency is of lesser importance than that of real-time risk evaluation which needs to be performed on demand in real-time in order to prevent users form be-

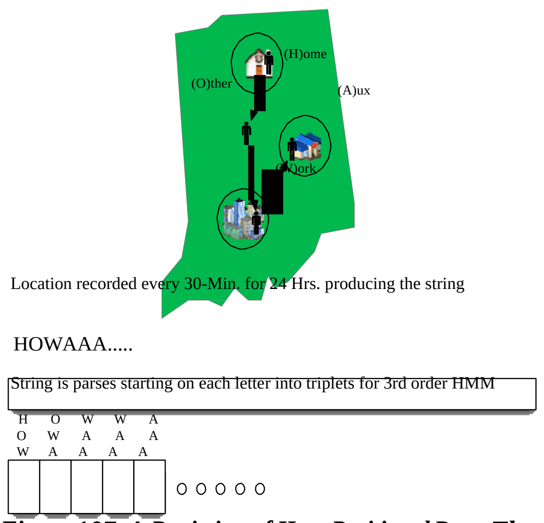
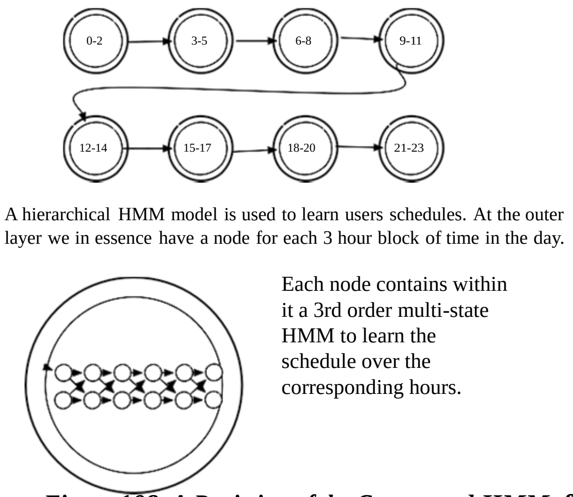

(H)ome
(O)ther
(A)ux
(W)ork
Location recorded every 30-Min. for 24 Hrs. producing the string
HOWAAA.....
String is parses starting on each letter into triplets for 3rd order HMM
H O W W A
O W A A A
W A A A A
o o o o o

Figure 107. A Depiction of How Positional Data Through the Day is Converted in to a String Over a Small Alphabet.

As previously mentioned, risk evaluation is based on the use of the forward algorithm. The forward algorithm runs in $O(n^2 \cdot t)$ where n is the number of states and t is the

number of time-blocks being analyzed; given an HMM M the forward algorithm returns the probability that a given sequence of positions $x_1, .., x_t$ is output by an HMM, given that it terminates in state σ_t. More formally, $\Pr[M \rightarrow x_1, ..., x_t | \sigma_t]$, for a given $x_1, ..., x_t$, and σ_t. However, for



A hierarchical HMM model is used to learn users schedules. At the outer layer we in essence have a node for each 3 hour block of time in the day.

Each node contains within it a 3rd order multi-state HMM to learn the schedule over the corresponding hours.

Figure 108. A Depiction of the Constructed HMM for Predicting Position.

risk analysis we have no preference for any specific termi- nal state, and so we are interested in $\Pr[M \rightarrow x_1, ..., x_t]$. A simple modification that sums the probabilities over all final states runs in $O(n^3 \cdot t)$, and returns the value of in- terest. Given the running time is cubic in the number of states and we need near real-time evaluations of the algo- rithm, we need to minimize the state space. To minimize the state space we actually construct 8 individual HMMs to learn patterns of behavior during different 3-hour periods of the day, and link them together through a simple state- machine.[2] The model is depicted in Fig. 3.

We justify this construction as a reasonable model because the risk of one's current geographic position is a function of both one's current position and recent historical position relative to the current time, as opposed to one's longterm schedule. We are currently in the process of experimentally determining the correct recent history window that will de- liver the best ability to detect abnormal behavior.

Temperature Temperature of the phone can be used to determine information relating to whether the phone is cur- rently in someone's physical possession. If the phone reads approximately body temperature ($37^o\ C$) then it is reason- able to assume that is in a person's possession.[3] Similarly, if the phone is at approximately room temperature or the outdoor ambient temperature, then the phone is likely ei- ther not directly on the person and is likely to have either

[2] This construction could be viewed as a Hierarchical HMM in which the transition distribution in the high-level HMM are all Kronecker δ-functions.

[3] There may need to be some invalidation of this metric at times when the ambient temperature is the same as body temperature.

been put down or remain in a bag.

While we believe there is strong potential to help use the phone's current temperature to monitor risks, our initial test of the Android phone is that the delay in converging to new temperatures by the phone's sensor makes this data unus- able for our intended applications. We found that when moving the phone in a pocket at body temperature and moving it onto a desk, it took on the order of tens of min- utes to converge to anywhere near the ambient room tem- perature. Further, in the same scenario it took several min- utes to decisively report non-body temperature readings.

Acceleration Acceleration measurements can be used in several manners to help determine risk. Techniques have been developed to measure a person's gait using the ac- celerometer in phones, assuming they are placed in an individual's pocket, or otherwise carried on the person [30, 15, 1]. While we do not intend to implement such a scheme ourselves, we are looking at the possibility of in- cluding the results of these works to deploy such a tech- nique in our larger sensor scheme. Further, we plan to use techniques that include simpler measurements but are based on other contexts. For example, if a user does *ex- plicitly* authenticate to the device, then at this point in time we know that the device is trusted. If the device stays in motion for the next several minutes, then one can assume that the correct user is still in possession of the device. In contrast if the phone becomes stationary for a prolonged period of time, the phone probably has been put down, and now alternative risk measurements must be used.

4.1.2. Social Networking Sensor Risk Measurement

One key aspect of our system is to use a form of social net- working for authentication and risk measurement. Imagine a scenario where a phone finds itself in a previously un- visited location, and other sensors are providing question- able risk data. However, imagine that the device can find the presence of a number of other phones that it frequently observes when in known low-risk states. The presence of these phones should indicate that the risk that an individual does not have proper possession of the phone is low: the phones of colleagues, friends and family members are near, so either the entire group is at risk (unlikely or the phone is simply in a new environment). Our system will employ a combination of white and black listing of other phones, which will alter the risk assessments made by the system. Additionally, we will learn "friendly" phones by determin- ing which other phones are frequently in the presence of the user in non-risky situations. This assessment will be done by considering both Bluetooth and 802.11 wireless networks.

Bluetooth. General Bluetooth frames are much more dif- ficult to detect than corresponding 802.11x frames *with the standard radio hardware built in to phones*.[4] There are two options to bypass this problem. The first is that the phones broadcast themselves in so called "Bluetooth dis- covery mode", this will make the phone visible to all, but can result in higher battery usage. The second is to pair specifically with those phones that are whitelisted to be considered friendly; pairing requires a one-time user inter- vention. In this case, the phones could attempt to pair when they are in close contact.

More problematically, our current implementation platform (Android v1.6) does not provide an API to interface with the Bluetooth infrastructure. Thus Bluetooth can only be accessed by the user, and not a risk-analysis program. An- droid (v2.0) does provide the implementation of such API, but there is currently no firmware upgrade for our reference platform (HTC G1 development).

WiFi (802.11x). Much of the widely deployed smart- phones allow their WiFi radios to operate in *promiscu- ous mode,* which permits the radio to listen to and com- municate the existence of frames that it can receive, even if the radio was not the target for the frame in question. This mode allows 802.11x radios to detect the presence of nearby devices. The only requirement to instantiate our social-networking risk measurement is to ensure that all the participating phones are broadcasting their position by sending beacons on regular intervals. It is yet to be deter- mined if the development platform supports such modes of operation.

4.1.3. Combining Risk Measurements.

A more sensitive risk measurement can be constructed if one does not require each sensor to independently gener- ate a risk metric in our risk model. However, in order to make our scheme flexible for different uses, and in devices with different subsets of sensors, we consider an archi- tecture that treats the sensor measurements independently, and then produces a global risk measurement. Note that this separation does not prevent the global risk measure- ment from learning co-dependencies between risk profiles of different sensors, and making use of such dependencies. There is a fair amount of research on methods for aggre- gating risk measurements in a number of different scenar- ios (e.g., Financial, Credit, Insurance, Intrusion Detection). Currently we are determining which, if any, of the current models provides a similar or appropriate model on which to base an aggregation of our sensor work. In the mean time,

[4]Relatively inexpensive hardware is available to capture general Blue- tooth packets, but it is not standard on known phones.

we use an expected value of the different risk metrics that is weighted with high-degrees to the positional and social networking schemes.

4.2. “Sensory Malware” threats and defenses

To fully understand the threat space of malware on smart- phones, we are exploring various attack scenarios. While traditional malware defenses focus on protecting resources on the computer (or as we would expect, on the smart- phone), we are specifically interested in the new class of at- tacks where *sensory malware* uses onboard sensors to steal information from the user’s physical environment [5]. For example, the user carries around a video and audio sen- sor (microphone) at all times, and thus immense amounts of information such as sensitive conversations, spoken passphrases or biometrics, keyboard acoustic emanations when placed next to a keyboard, and broader surveillance becomes possible. Video “sensors” can gather visual infor- mation about a user’s private environment such as pictures of colleagues [38], which may be sensitive with military and intelligence-gathering agencies. Accelerometers and GPS sensor information can be used to infer location and activity patterns of users such as soldiers, thus compromis- ing military secrecy.

While generic architectures [10, 23] have been proposed to control access to the network, for example, after soft- ware has accessed certain sensor information, various vec- tors exist for leaking garnered information. Overt channels between components on the smartphone (Android provides very little security against communicating applications, for example), or covert channels between related malware ap- plications (through a storage channel, for example) are cur- rently viable vectors for leaking sensitive data to adver- saries. It is even possible to leverage other “blessed” ap- plications on the phone to act as a carrier for such informa- tion (by invoking a web-browser with an encoded URL, for example). Thus we are interested in building a unified ar- chitecture for controlling access to sensor data, and limiting what information can be gleaned from the user’s environ- ment unless he or she is making use of legitimate appli- cations. We are currently building a software prototype of one instance of sensory malware to demonstrate the reality of the threat, and to better understand defensive techniques to limit such malware.

We aim to study types of sensory malware that are stealthy and thus use few resources on the mobile device. For ex- ample, speech-based malware may use several heuristics to target analysis at only specific portions of the audio sample. Such targeted analysis can drastically reduce the amount of resources needed to analyze audio samples, thus decreasing the observability of such malware. To conserve power, such malware can also target its offline processing to when the mobile device is connected to a power source for charging. Under such circumstances the malware uses few precious resources and does not detract from the user’s experience. Speech malware of this type may even operate using more general “profiles” that tune the malware to recognize sev- eral different situations, or contexts, such as a recognized phone number that is dialed. Based on the context, the speech malware can, for example, detect a credit card cus- tomer service line and target analysis to credit card number extraction. Calls to financial institutions such as banks of- ten require portions of the user’s social security number, which could be extracted similarly. Such profiles can make use of other clues such as audio or video triggers to better target surveillance and transmit specific information.

To counter such threats, therefore, we need a framework that is better equipped to deal with sensory malware threats. Research is needed to understand the threat space of sen- sory malware, so that effective defenses can be deployed. As mentioned earlier, existing solutions are unable to deal with situations in which malware communicates through covert channels, and thus such work must also take into ac- count anomalous resource usage to detect such covert chan- nels. Being low-powered devices makes the job of defen- sive software much more challenging, and thus lightweight detection techniques are necessary. It is even possible that the mobile platform can leverage computation in the cloud for “outsourced intrusion detection,” which might strike a tradeoff between the time to detection and power consump- tion.

4.3. Side-channel detection and mitigation

It is well known that the contents of encrypted traffic can be disclosed by its attributes observable to a eavesdrop- per, for example, packet sizes, sequences, inter-packet tim- ings. Such attributes, often referred to as side-channel in- formation, often pose a grave threat to the confidential- ity of the communication under the protection of cryp- tographic protocols. Side-channel leaks have been ex- tensively studied for decades, in the context of secure shell (SSH) [27], video-streaming [26], voice-over-IP (VoIP) [37], web browsing and others. As an example, a line of research conducted by various research groups stud- ied anonymity issues in encrypted web traffic. It has been shown that because each web page has a distinct size, and usually loads some resource objects (e.g., images) of differ- ent sizes, the attacker can fingerprint the page so that even when a user visits it through HTTPS, the page can still be re-identified [9, 29]. This vulnerability is known to be a serious concern for anonymity channels such as Tor [31], which are expected to hide users' page-visits from eaves-droppers.

A sensor grid system can also be highly susceptible to the threat of side-channel leaks. As described before, such a system collects data through distributed sensors, processes it within a cloud, and delivers the data and related services to end clients. This highly distributed computing paradigm is fraught with the hazards of information leaks, when con- fidential data are transmitted between the sensors and the cloud, and between the cloud and the clients, despite the protection of the state-of-the-art cryptographic techniques. Such privacy risks are described as follows:

Wireless Sensor Communication. The wireless channel connecting the sensors to the cloud is extremely vulnerable to the eavesdropping attack. The sensitive data delivered through this channel can be easily intercepted and analyzed by the adversary. Though encryption can prevent a direct disclosure of the data, it does not cover the side-channel in- formation, which, under some circumstances, can be used to infer the content of the sensitive data. As an example, collaborating with Microsoft Research (MSR), we recently discovered that even for the organization deploying up-to- date WPA/WPA2 Wi-Fi encryptions, it cannot prevent an unauthorized party from collecting the query words its em- ployees enter into Google/Yahoo/Bing Search. This is be- cause the suggestion-list features of these search engines makes the sizes of the packets generated in response to different query letters distinct. As a result, the adversary who observes these packets, despite not gaining access to their contents, can map their sizes to the different letters one types into the search engines.

Cloud–consumer Communication. The encrypted data exchanged between the cloud and its customers are equally subject to the side-channel threat. Cloud computing is built upon the infrastructure of *software as a service* (SaaS), through which *web applications* are delivered as services to web clients. Unlike its desktop counterpart, a web applica- tion is split into browser-side and server-side components. As a result, a subset of its internal information flows (i.e., data flows and control flows) are inevitably exposed on the network, which reveal application states and state transi- tions. Our collaborative research with MSR reveals that the side-channel weakness of SaaS is fundamental, which can be used to infer a large amount of information from many high-profile, extremely popular web applications. The sen- sor grid system also faces the same threat: it offers services and data to its customers through web applications, whose side-channel information could lead to the disclosure of the data, even when the communication has been protected by the cryptographic protocols like HTTPS.

The seriousness of the side-channel threat varies from case to case, depending on the features of the data and the way in which they are transmitted. An important research, there- fore, becomes how to design a systematic way to detect the side-channel vulnerabilities within sensor/cloud inter- actions and the web applications that serve the sensor grid's customers. A possible solution is to use *information-flow analysis* [28], when the source code of related software is available. The software developer can first label taint sources within a program, e.g., variables that contain sen- sitive user data, and then run a detection tool to analyze its source code and track the propagation of taint data through both data flows and control flows. Whenever taint data are found to be transmitted across the network between the ap- plication's client and server components, an information- leak evaluation is performed to understand whether side- channel information, such as packet sizes, sequences and timings, can be linked back to the content of the data. When the source code is unavailable, we can use the techniques like fuzz testing to evaluate sensor-cloud interactions and cloud-client interactions on different data sets, to identify the correlation between the attributes of encrypted traffic and the content of the data.
Control of side-channel leaks can also be highly nontriv- ial, particularly when web applications are involved. Our collaborative research with MSR reveals that conventional defenses like packet padding and adding noise can be less effective and more costly than expected, without con- sidering the specific properties of individual applications. This problem comes from the difficulty in hiding the side- channel information related to state transitions specific to each application, and the limited information an application has about the attributes of the web traffic it generates, due to the extension or compression made by the web server. This vulnerability calls for a change in the current way of developing web applications to include the collaborations among multiple related parties: as an example, we could let the software developer specify the policies for padding packets at different program states, and the web-server ven- dor enforce the policies within the web server that actually generates the packets.

5. RELATED WORK

Kapadia et al. [16] list several security challenges for sim- ilar smartphone based sensing environments. While their work focuses mainly on an opportunistic sensing model where sensors are *tasked* for readings sent back as *reports* to other users or applications in urban sensing environ- ments, we focus on environments where sensors push mas- sive amounts of data to a compute cloud. We now list re- lated work for the three specific problems discussed in Sec- tion 3..

5.1. Mobile phone security and privacy

There has been some work in using sensors to establish context for different purposes on smartphones. The work of Peddemors et al. [24] uses past networking and sensor events to predict future network events. They give exam- ples of predicting network availability. The ability to pre- dict events is distinct from deviating from normal or pre- scribed behavior. Nonetheless they use the prediction of being at home or work, and for durations. Therefore, the system should be considered. Of particular problem is the complexity of computing predicted events, which would be too slow in our scenario.

The work of Tanviruzzaman et al. [30] is most similar to that discussed here. In their work, they suggest the use of a hierarchy of sensor information to establish authentica- tion, and show some work on using accelerometer data on an iPhone to produce a biometric that can be used to au- thenticate to the phone.

Other work by Jong-Kwon and Hou [18] has predicted user behavior and movements from the perspective of a large WiFi network, for the purposes of assigning scarce resources appropriately. However, we do not rely on one overarching network for our positioning system. Yet, the possibility exists that such work could be used to have the network aid in performing risk analysis.

The field of smartphone security and the security of cellphone infrastructure is now being widely researched. Traynor [32] gives a short overview of infrastructure pos- sibilities and problems. Traynor et al. [34] consider the potential effect of a malnet of smartphones on the cellular network's infrastructure. Enck et al. [13] discuss exploits in the SMS-network infrastructure, and Traynor et al. [33] discuss mitigation strategies for such exploits.

Relating to mobile phone security, there has been recent in- terest in maintaining their security. The potential to attack these devices, and that they would suffer similar security fates to personal computers, such as viruses and malware, has been long understood [8]. Specific approaches to con- sidering defense against such software on smartphones has been considered by Cheng et al. [7]. The specific strengths and weaknesses of the Android security model are explored by Ongtang et al. [23]. The ability to securely determine if software downloads are trusted on such devices is explored by Enck et al.[11]. Enck et al. [12] give an introduction to understanding the Android security model specific to the smartphones we are using for implementation.

5.2. Sensory malware threats and defenses

As mentioned earlier, researchers are already investigating attacks and defenses related to sensory malware [5]. Xu et al. [38] provide a proof-of-concept implementation of video-capture malware. Their malware captures video and transmits this video after suitable compression to lessen the burden on the network. These malware do not appear to be stealthy enough because of the large amounts of video data transferred on the network. We thus seek to develop and evaluate solutions where malware is even more stealthy, by limiting the network communication. In fact, we would like to study situations where network access is limited com- pletely using techniques such as Kirin, a lightweight secu- rity certification mechanism for applications on Android. Even in cases where a system such as Saints [23] is used to control the interaction between applications, we would like to study the use of covert channels to circumvent such mechanisms.

Detection techniques such as behavioral detection of mal- ware by monitoring system calls [3], and power consump- tion [20] already attempt to detect malware on mobile plat- forms. We aim to study the limits of such detection tech- niques since resources are limited, and how malware can circumvent detection because of the inherent limitations on the detection techniques.

5.3. Side-channel information leaks

Side-channel leaks have been known for decades: a doc- umented attack has been dated back to 1943 [22]. The threat has been extensively studied in different contexts: information is found to be exposed through electromag- netic signals (e.g., keystroke emanation [35]), shared mem- ory/registers/files between processes (e.g., the recent dis- covery of the side-channel weakness in Linux process file systems [39]), CPU usage metrics, etc. Recently, such in- formation leaks are found to threaten cloud computing plat- forms like Amazon EC2 [25].

Encrypted communications are often subject to the side-channel attacks, which leverage such information as packet timings and sizes to infer the contents of encrypted data. Prominent examples include Brumley et al.'s attack on the RSA secret keys used in OpenSSL [4], Song et al.'s work on keystroke inference from SSH [27], Wright et al. and others' analysis of phrases and sentences from the variable- bit-rate encoding in VoIP [37], and Saponas et al.'s detec- tion of movie titles in an encrypted video-streaming system (Slingbox Pro) [26]. Encrypted web communication has also been found to be vulnerable to the side-channel attack. Prior research shows that a network eavesdropper can often fingerprint web pages using their side-channel characteris- tics to identify the pages the victim visits. This idea first appeared in the personal communication among Wagner, Schneier and Yee in 1996 [36], and was later demonstrated in a course project report in 1998 by Cheng et al. [6]. Sun et al. [29] and Danezis [9] both indicated the impacts of the attack on anonymity channels like Tor, MixMaster and WebMixes. It was also discussed by Bissias et al. [2], who studied WPA and IPSec, instead of SSL/TLS in other re- search.

6. SUMMARY

We have outlined a high-level architecture that should both be realizable, and provide for the ability to perform on- demand analysis and processing of data from a large num- ber of heterogeneous and globally placed sensors. The net- work is structured so that it is feasible to consider real or near-real time processing and interpretation of the data with appropriate resources. However, challenges remain in de- termining how to assure privacy, integrity and provenance of the data from its collection, through its life-cycle of pro- cessing to final consumption. The authors' belief is that the largest research questions based on our model lie at the tail ends of the data life-cycle; namely, there are open research questions at data-collection by smartphone sensors and in the final delivery of a processed data-consumable. Specific directions aimed at solving these problems have been dis- cussed, along with initial development of solutions. We summarize this in Table 1

Table 1. Summary of the Three Threats, Associated Dangers and Mitigation Strategies We Actively Address.

Threat	Danger	Mitigation
Sensor Ab-duction	Malicious Sensor Data	Detection of non-regular usage
Side-channel information leakage	Communication en-cryption is circum-vented by analysis of packet sizes& spac-ing	Flow-analysis and padding
Sensor mal-ware	Sensor data theft	Sensor access control mod-els

REFERENCES

[1] *Identifying users of portable devices from gait pattern with accelerometers*, volume 2, 2005.

[2] G.D. Bissias, M. Liberatore, D. Jensen, and B.N. Levine. "Privacy vulnerabilities in encrypted http streams," In proceedings of Privacy Enhancing Technologies Workshop (PET 2005), pages 1–11, 2005.

[3] A. Bose, X. Hu, K.G. Shin, and T. Park. "Behavioral detection of malware on mobile handsets," In MobiSys '08: Proceeding of the 6th international conference on Mobile sys- tems, applications, and services, pages 225–238, New York, NY, USA, 2008. ACM.

[4] D. Brumley and D. Boneh. "Remote timing attacks are practical," In proceedings of the 12th USENIX Security Symposium, pages 1–14, 2003.

[5] L. Cai, S. Machiraju, and H. Chen. "Defending against sensor-sniffing attacks on mobile phones," In MobiHeld '09: proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds, pages 31–36, New York, NY, USA, 2009. ACM.

[6] H. Cheng and R. Avnur. "Traffic analysis of SSL encrypted web browsing," http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.1201\&rep=rep1\&type=url\&i=0, 1998.

[7] J. Cheng, S.H.Y. Wong, H. Yang, and S. Lu. "Smartsiren: virus detection and alert for smartphones," In MobiSys '07: proceedings of the 5th international conference on Mobile systems, applications and services, pages 258–271, New York, NY, USA, 2007. ACM.

[8] D. Dagon, T. Martin, and T. Starner. "Mobile phones as computing devices: The viruses are coming!" *IEEE Pervasive Computing*, 3(4):11–15, 2004.

[9] G. Danezis. "Traffic analysis of the http protocol over TLS," http://homes.esat.kuleuven.be/~gdanezis/TLSanon.pdf, as of Dec 2009.

[10] W. Enck, M. Ongtang, and P. McDaniel. "On lightweight mobile phone application certification," In CCS '09: proceedings of the 16th ACM conference on Computer and communications security, pages 235–245, New York, NY, USA, 2009. ACM.

[11] W. Enck, M. Ongtang, and P. McDaniel. "On lightweight mobile phone application certification," In CCS '09: proceedings of the 16th ACM conference on Computer and communications security, pages 235–245, New York, NY, USA, 2009. ACM.

[12] W. Enck, M. Ongtang, and P.D. McDaniel. "Understand- ing android security," *IEEE Security & Privacy*, 7(1):50–57, 2009.

[13] W. Enck, P. Traynor, P. McDaniel, and T. La Porta. "Exploit- ing open functionality in sms-capable cellular networks," In CCS '05: proceedings of the 12th ACM conference on Com- puter and communications security, pages 393–404, New York, NY, USA, 2005. ACM.

[14] K. Farrahi and D.G. Perez. "Learning and predicting multimodal daily life patterns from cell phones," In J.L. Crowley, Y. Ivanov, C.R. Wren, D. Gatica-Perez, M. Johnston, and R. Stiefelhagen, editors, ICMI, pages 277–280. ACM, 2009.

[15] T. Iso and K. Yamazaki. "Gait analyzer based on a cell phone with a single three-axis accelerometer," In MobileHCI '06: proceedings of the 8th conference on Human-computer interaction with mobile devices and services, pages 141–144, New York, NY, USA, 2006. ACM.

[16] A. Kapadia, D. Kotz, and N. Triandopoulos. "Opportunis- tic Sensing: Security Challenges for the New Paradigm," In The First International Conference on Communication Systems and Networks (COMSNETS), January 2009.

[17] C. Karlof, N. Sastry, and D. Wagner. "Tinysec: a link layer security architecture for wireless sensor networks," In Sen- Sys '04: proceedings of the 2nd international conference on Embedded networked sensor systems, pages 162–175, New York, NY, USA, 2004. ACM.

[18] J.K. Lee and J.C. Hou. "Modeling steady-state and tran- sient behaviors of user mobility: formulation, analysis, and application," In MobiHoc '06: proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing, pages 85–96, New York, NY, USA, 2006. ACM.

[19] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D.
Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. "TinyOS: An operating system for sensor networks," In Ambient Intelligence. Springer Verlag, 2004.

[20] L. Liu, G. Yan, X. Zhang, and S. Chen. "Virusmeter: Preventing your cellphone from spies," In E. Kirda, S. Jha, and D. Balzarotti, editors, *RAID*, volume 5758 of Lecture Notes in Computer Science, pages 244–264. Springer, 2009.

[21] M. Luk, G. Mezzour, A. Perrig, and V. Gligor. "Minisec: a secure sensor network communication architecture," In IPSN '07: proceedings of the 6th international conference on Information processing in sensor networks, pages 479–488, New York, NY, USA, 2007. ACM.

[22] Wired News. "Declassified NSA document reveals the secret history of tempest," http://www.wired.com/ threatlevel/2008/04/nsa-releases-se.

[23] M. Ongtang, S. E. McLaughlin, W. Enck, and P. D. Mc- Daniel. "Semantically rich application-centric security in Android," In ACSAC, pages 340–349. IEEE Computer Society,
2009.

[24] A. Peddemors, H. Eertink, and I. Niemegeers. "Predicting mobility events on personal devices", *Pervasive and Mobile Computing, Special issue on Human Behaviour in Ubiquitous Environments,* To Appear.

[25] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," In CCS '09: proceedings of the 16th ACM conference on Computer and communications security, pages 199–212, New York, NY, USA, 2009. ACM.

[26] T.S. Saponas, J. Lester, C. Hartung, S. Agarwal, and T. Kohno. "Devices that tell on you: privacy trends in consumer ubiquitous computing," In SS'07: proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, pages 1–16, Berkeley, CA, USA, 2007. USENIX Association.

[27] D.X. Song, D. Wagner, and X. Tian. "Timing analysis of keystrokes and timing attacks on SSH," In SSYM'01: proceedings of the 10th conference on USENIX Security Symposium, pages 25–25, Berkeley, CA, USA, 2001. USENIX Association.

Appendix B

Overview of Status of Clouds

1. Introduction

The importance of simulation is well established with large programs, especially in Europe, USA, Japan and China supporting it in a variety of academic and government initiatives. The requirements and consequent architecture of large scale supercomputers is well understood although there are important challenges in meeting performance goals seen by international drives to reach first petascale (starting 15 years ago) and now exascale performance. Performance on closely coupled parallel simulations drives both hardware (low latency high bandwidth networks, high flop CPU's) and software that can exploit it. Grids covered both the linkage of such computers and broader computing facilities. This has spurred rise in high throughput computing, workflow and service oriented architectures (Software as a service); concepts of lasting value. Major data intensive applications like LHC data analysis highlighted the many important pleasingly parallel applications that these were a major driver of Grid and many task systems. Now the strong commercial interest is driving clouds and we can ask how they fit in? Clouds offer on-demand service (elasticity), economies of scale from sharing, a plethora of new jobs making clouds attractive for students & curricula and several challenges including security. Clouds lie in between grids and HPC supercomputers in their synchronization costs so all the high throughput jobs run on grids should perform well on clouds. In this paper, we suggest that there is a class of explicitly parallel jobs that do not need the highest performance interconnect and will have good performance and good user experience on clouds. We describe this in an application analysis in section2. Of course, HPC supercomputers can do "all applications" subject to reservations about limited I/O (disk) capabilities. However, they are overkill for many problems and it seems better to reserve such machines for the high-end applications that require them and use commodity cloud environments when appropriate.

We stress that clouds offer not just a new humongous data center architecture but striking new software models spurred by the competitive Platform as a Service PaaS market. In section 3 we focus on the possibilities suggested by MapReduce.

The term cloud is being in many ways so let's first define a public data center model that describes the major offerings of Microsoft, Amazon and Google. Their data centers are composed of containers of racks of servers which number between 10,000 and a million. Each server has 8 or more cpu cores and around 64GB of shared memory and one or more terabyte local disk drives. GPUs or other accelerators are not common. There is a network that allows messages to be routed between any two servers, but the bisection bandwidth of the network is very low and the network protocols implement the full TCP/IP stack so that every server can be a full Internet host with optimized traffic between users on the Internet and the servers in the cloud. In contrast supercomputer networks minimize interprocessor latency and maximize bisection bandwidth. Application data communications on a supercomputer generally take place over specialized physical and data link layers of the network and interoperation with the Internet is usually very limited.

2. A Cloud Defined

Each server in the data center is host to one or more virtual machines and the cloud runs a “fabric controller” which manages large sets of VMs fort scheduling and fault tolerance across the servers and acts as the operating system for the data center. An application running on the data center consists of one or more complete VM instances that implement a web service. The basic unit of scheduling involves the deployment of one or more entire operating systems, which is much slower than installing and starting an application on a running OS. Most large scale cloud services are intended to run 24x7, so this long start-up time is negligiblen although running a “batch” application on a large number of servers can be very inefficient because of the long time it may take to deploy all the needed VMs. Data in a data center is stored and distributed over many spinning disks in the cloud servers. This is a very different model than found in a large supercomputer, where data is stored in network attached storage. Local disks on the servers of supercomputers are not frequently used for data storage.

There are more types of clouds than is described by this public data center model. For example, to address a technical computing market, Amazon has introduced a specialized HPC cloud that uses a network with full bisection bandwidth and supports GPGPUs. The major commercial clouds offer higher level capabilities -- commonly termed Platform as a Service PaaS – built on a basic scalable IaaS Infrastructure as a Service. For technical computing, important platform components include tables, queues, database, monitoring, roles (Azure), and the cloud characteristic of elasticity (automatic scaling). MapReduce, which is discussed below, is another major platform service offered by these clouds. Currently the different clouds have different platforms although the Azure and Amazon platforms have many similarities. The Google Platform is targeted at scalable web applications and not as broadly used in technical computing community as Amazon or Azure, but it has been used on some very impressive projects. We expect more academic interest in PaaS as the value of platform capabilities become clearer.

“Private clouds” are small dedicated data centers that have various combinations of the properties above and typically use one of the four major open source (academic) cloud environments Eucalyptus, Nimbus, OpenStack and OpenNebula (Europe) which focus at the IaaS level with interfaces similar to Amazon. FutureGrid is an NSF research testbed for cloud technologies and it operates a grid of cloud deployments running on modest sized server clusters with support for all four academic IaaS. Private clouds do not fully support the interesting platform features of commercial clouds. Open source Hadoop and Twister offer MapReduce features similar to those on commercial cloud platforms and there are open source possibilities for platform features like queues (RabbitMQ, ActiveMQ) and distributed data management system (Apache Cassandra). However, there is no complete packaging of PaaS features available today for academic or private clouds. Thus interoperability between private and commercial clouds is currently only at IaaS level where it is possible to reconfigure images between the different virtualization choices and there is an active cloud standards activity. The major commercial virtualization products such as VMware and Hyper-V are also important for private clouds but also do not have built-in PaaS capabilities.

3. Mapping Applications to Clouds

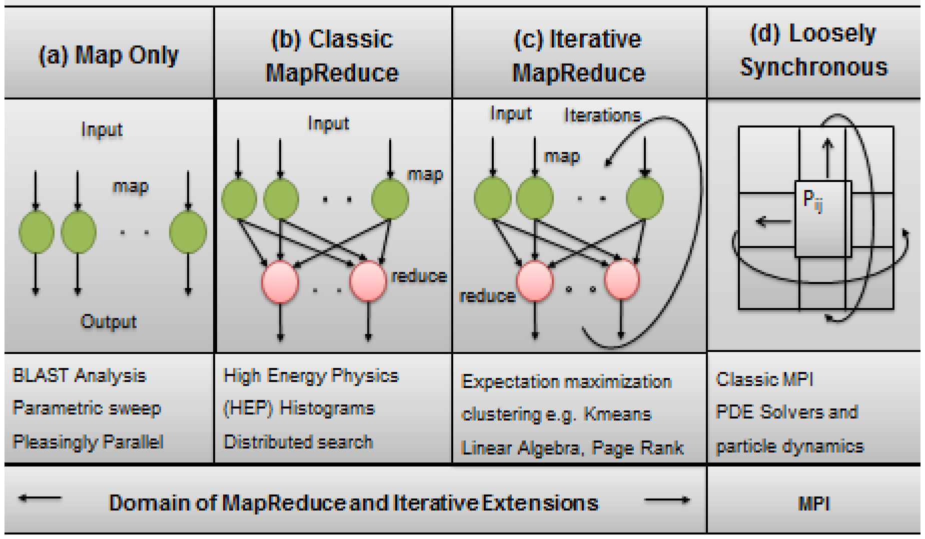


Figure 109: Forms of Parallelism and their application on Clouds and

Previously we discussed mapping applications to different hardware and software in terms of 5 "Application Architectures"[1] mainly aimed at simulations and extended it to data intensive computing [2, 3]. One category, synchronous, was popular 20 years ago but is no longer significant. It describes applications that can be parallelized with each decomposed unit running the identical machine instruction at each time. Another category, asynchronous is typically not important in practical computational science and engineering. There was also a category of metaproblems, which describe the domain supported by workflow with coarse grain interlinked components. The other categories were pleasingly parallel (essentially independent) and loosely (bulk) synchronous which are critical application classes that possibly combined in metaproblems describe the bulk of eScience. As mentioned above, pleasingly parallel problems whether parameter searches for simulations or analysis of independent data chunks (as in LHC events) are very suitable for clouds. Loosely synchronous problems include partial differential equation solution and particle dynamics and after parallelization, consist of a succession of compute-communication phases.

Clouds naturally exploit parallelism from multiple users or usages. The Internet of things will drive many applications of the cloud. It is projected that there will soon be 50 billion devices on the Internet. Most will be small sensors that send streams of information into the cloud where it will be processed and integrated with other streams and turned into knowledge that will help our lives in a million small and big ways. It is not unreasonable for us to believe that we will each have our own cloud-based personal agent that monitors all of the data about our life and anticipates our needs 24x7. The cloud will become increasing important as a controller of and resource provider for the Internet of Things. As well as today's use for smart phone and gaming console support, "smart homes" and "ubiquitous cities" and the current AFRL project build on this vision. We expect a growth in these areas with emergence of cloud supported/controlled robotics.

Looking at data intensive applications we can re-examine the pleasingly parallel and loosely synchronous category as shown in figure 1 above. This introduces map-only (identical to pleasing parallel), and separates off MapReduce and Iterative MapReduce classes from the large loosely synchronous class whose remaining members are the last sub category d) on the right of figure 1. This area requires HPC architectures with low latency high bandwidth interconnect. The MapReduce class b) consists of a single map (compute) phase followed by a reduction phase such as gathering together the results of queries following an Internet search or LHC data analysis (histogram) of different datasets. As implemented in Hadoop, one would normally communicate between Map and Reduce phases by writing and reading files. This leads to excellent fault tolerance and dynamic scheduling features. At SC11, there was some buzz in favor of data analytics and Hadoop but that this is not clearly reasonable as many data analysis (mining) applications involve kernels that do not fit Map only or MapReduce categories. Many algorithms including those with linear algebra (needing to be parallelized) fall into the category c) Iterative MapReduce in figure 1. Problems in this category consist of multiple (iterated) Map phases followed by reduction or collective operation communication phases. They do not have the many local communication messages typically needed in parallel simulations shown in fig 1d) but rather larger collective operations mixing compute and communication. We do not expect traditional MapReduce to be broadly useful but the Iterative extension is much more promising but the breadth of its applicability needs much more study. Iterative MapReduce is a programming model that can have the performance of MPI and the fault tolerance and dynamic flexibility of the original MapReduce. Open source Java Twister[4, 5] and Twister4Azure[6, 7] have been released as an Iterative MapReduce framework. Figure 2 compares Twister4Azure with Amazon and a classic HPC configuration on a map-only case while figure 3 shows Azure4Twister having a smooth execution structure and modest communication overhead (the uncolored gaps) on a parallel data analytics algorithm. We expect the commonly used expectation maximization

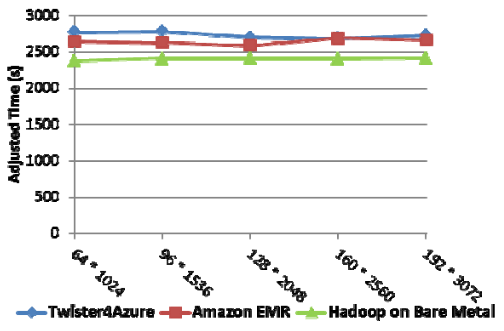


Figure 110: A Map Only example pairs sequence

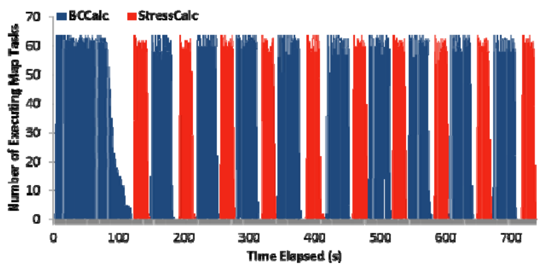


Figure 111: Parallel MDS on Azure4Twister showing communication (white) and two compute map phases

(EM) approach used for example in Multidimensional Scaling MDS application of fig 3, to be particularly attractive for iterative MapReduce as EM can have large compute/communication ratios. Category c) extends the clear value of clouds in the categories a) and b) of figure 1.

3. CLOUDS AND REPOSITORIES

It is traditional to set up data repositories for large observational projects. Examples are EOSDIS (Earth Observation), GenBank (Genomics), NSIDC (Polar science), and IPAC (Infrared astronomy). The fourth paradigm implies an increase in data mining (analytics) based on such data and this implies repositories need computing as well as data. We also expect that one should bring the computing to the data and not vice versa. Thus we do not expect researchers to download large petabyte data samples to their local cluster; rather we expect repositories to be associated with cloud resources (as cheapest and elastic) that allow data analytics on demand. Again further work is needed here. Some questions include the data storage architecture (database or NOSQL) and how one supports mining of multidisciplinary science involving data from different fields stored in different clouds.

4. Cloud Research Issues

We list areas where is substantial research activity and where we can expect major changes.

- New applications such as Biomedical and bioinformatics applications where cloud architecture brings special challenges in the area of privacy (see later). Furthermore, Clouds have been attractive platforms for these applications as they are emerging big data areas and there is less history in using existing platforms.
- Sensor webs studied in this project are another emerging area where elastic nature of Clouds is well suited for the often bursty nature of sensor data.
- Big data applications based on new MapReduce or Iterative MapReduce environments are attractive on Clouds and result in broad research areas include addressing both programming and storage challenges. Latter include SQL and NOSQL models and the reconciliation of distributed data and centralized cloud computing
- Scheduling models optimized for MapReduce and for other Cloud usage modes such as scalable sensor webs (Sensor Grids or Clouds) where one has Clouds controlling and supporting a distributed Grid of sensors.
- Optimizing the run time features and performance for MapReduce and Iterative MapReduce. This includes new reduction primitives, polymorphic implementation on different systems with for example, exploitation of high performance networks as in classic MPI research.
- Support of federation of clouds and cloud bursting (typically the linkage of private and public Clouds) and on-demand cloud federation.
- New storage models such as data parallel HDFS and Hbase (Bigtable).
- NOSQL table structures such as Cassandra and commercial approaches such as Amazon SimpleDB and Azure Table.
- Economic models for an ecosystem with multiple cloud systems and CI.
- Research on Cloud software stacks. There is research at all levels of the software stack with two rather different emphasis areas. Research on systems that provide basic virtual machine provisioning, deployment and management. This includes Eucalyptus, Nimbus, OpenStack and OpenNebula with virtual networking as a distinct activity. At the other end are integration of capabilities to provide rich Platform-as-a-Service as offered by major commercial systems. Concepts such as appliances provide novel ways of delivering these capabilities.
- Clouds tend to achieve scalability by allowing faults. Research is needed on both, how to expose faults to users as well as services to build fault tolerant applications. Most research in HPC tends to be

on forbidding faults; however Clouds highlight a different philosophy with resilient applications running on faulty systems.

- Green IT is naturally synergistic with Clouds and related research includes examining the impact of Cloud features on power use, including the cost of powering idle machines supporting elastic clouds as well as a application aware approaches to power management.

Security policies and mechanisms: Clouds tend to emphasis the need for quality security mechanisms due to the sharing of storage and computing. One research area investigates hybrid architectures with algorithms broken into two; a low cost but non privacy preserving part running on an intrinsically secure private clouds, and a time consuming but privacy preserving part executing on a public cloud. Genomic data (human) and other health records are demanding here. The concept of differential privacy and health data anonymization is an active research topic. As well as basic security for computing and storage there is research on privacy preserving search with the elegant but time consuming concept of Homomorphic Encryption which allows encrypted data to be searched by encrypted queries.

Standards: There are many important standard activities, from those specifying the basic virtual machine structure to higher-level standards defining the PaaS environment, for example, queue and table structures. Although there is some support for these standards – such as OCCI (from OGF) in OpenNebula and OpenStack – this area is still under development. NIST and IEEE are playing leadership roles.

5. References

1) Fox, G.C., R.D. Williams, and P.C. Messina, *Parallel computing works!* 1994: Morgan Kaufmann Publishers,
2) calculating all Jaliya Ekanayake, Thilina Gunarathne, Judy Qiu, Geoffrey Fox, Scott Beason, Jong Youl Choi, Yang Ruan, Seung-Hee Bae, and Hui Li, *Applicability of DryadLINQ to Scientific Applications*. January 30, 2010, Community Grids Laboratory, Indiana University.
3) Judy Qiu, Jaliya Ekanayake, Thilina Gunarathne, Jong Youl Choi, Seung-Hee Bae, Yang Ruan, Saliya Ekanayake, Stephen Wu, Scott Beason, Geoffrey Fox, Mina Rho, and H. Tang, *Data Intensive Computing for Bioinformatics*. December 29, 2009.
4) SALSA Group. *Iterative MapReduce*. 2010 [accessed 2010 November 7]; Twister Home Page Available from: http://www.iterativemapreduce.org/.
5) J.Ekanayake, H.Li, B.Zhang, T.Gunarathne, S.Bae, J.Qiu, and G.Fox, *Twister: A Runtime for iterative MapReduce*, in *Proceedings of the First International Workshop on MapReduce and its Applications of ACM HPDC 2010 conference June 20-25, 2010*. 2010, ACM. Chicago, Illinois.
6) *Twister for Azure*. [accessed 2011 May 21]; Available from: http://salsahpc.indiana.edu/twister4azure/.
7) Thilina Gunarathne, Bingjing Zhang, Tak-Lon Wu, and Judy Qiu, *Portable Parallel Programming on Cloud and HPC: Scientific Applications of Twister4Azure*, in *IEEE/ACM International Conference on Utility and Cloud Computing UCC 2011*. December 5-7, 2011. Melbourne Australia. http://www.cs.indiana.edu/~xqiu/scientific_applications_of_twister4azure_ucc_17_4.