

Final Report
AFRL SBIR Phase 2 Project
FA8650-06-C-4404

**Anabas Sensor-Centric Grid of Grids Middleware
Management System**

September 24, 2008

Notice and Signature Page

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY)		2. REPORT TYPE Final Report (Draft)		3. DATES COVERED (From - To) 6/19/06 - 6/18/08	
4. TITLE AND SUBTITLE Grid of Grids for Information Management				5a. CONTRACT NUMBER FA8650-06-C-4404	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Alex Ho, Geoffrey Fox				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Anabas, Inc. 580 California Street Suite 1600 San Francisco CA 94104				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT A major challenge faces the United States (US) warfighter as we enter the 21 st century. The very composition and nature of the war fighting operation required to win a war has shifted from being dependent on industrial technologies to information technologies. The solutions to these challenges are forcing the warfighter to assimilate and process vast amounts of information from a broad spectrum of domains. The Global Information Grid (GiG) will provide the global connectivity in the net-centric environment for the information to get to where it's needed. However the information will still need to be processed and converted to knowledge to empower the warfighter to make the right decisions and succeed. This project focuses on developing the Net-Centric Collaboration Grid Middleware, and in particular Sensor-Centric Grid Middleware, that is needed to build domain specific warfighter command & control (C2), decision support solutions within a sensor-centric grid-enabled environment.					
15. SUBJECT TERMS Net-centric, Sensor-centric, Grid of Grids, Collaborative, UDOP, COP, Global Information Grid					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT No	18. NUMBER OF PAGES 282	19a. NAME OF RESPONSIBLE PERSON Alex Ho
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code) 415-637-4198

List of Figures

Figure 2-1 Integrating modeling and simulation systems with real-time and archived GPS sensor streams into Sensor-Centric Grid of Grids	5
Figure 2-2 A typical variance pattern, observed for a ‘talking head’ video segments.	9
Figure 2-3 Region detection on a typical video zooming and panning sequence.	10
Figure 2-4 Region detection on another typical video panning sequence.	11
Figure 2-5 Fast changing region detection in an Impromptu client.	12
Figure 2-6 A sample template grid design	14
Figure 2-7 An Eclipse-based Grid Builder Tool Layout Design	15
Figure 2-8 Resource management architecture in Grid Builder	16
Figure 2-9 An NCCGBT supported EMF model view of BPEL-extended elements	19
Figure 2-10 A GEF-based editor for Grid Builder	20
Figure 2-11 An updated design for Grid Builder to support ServoGrid/Quakesim	22
Figure 2-12 An enhanced BPEL editor for Grid Builder	23
Figure 2-13 An enhanced, distributed resources viewing interface for Grid Builder	25
Figure 2-14 A distributed grid service management architecture	26
Figure 2-15 A high-level Grid Builder architecture	28
Figure 2-16 A bootstrap service interface for the Grid services management system	29
Figure 2-17 A management interface for managing a Narada Brokering fabric	30
Figure 2-18 An illustration of the QuakeSim2 portal interface	34
Figure 3-1 A conceptual any time, anywhere, anything Grid of Grids system	35
Figure 3-2 SCGMMS overall architecture	37
Figure 3-3 Structure of A Sensor Client Program	40
Figure 3-4 Computational Service	40
Figure 3-5 An overview of the Grid Builder architecture	42
Figure 3-6 Class Diagram of Grid Builder	44
Figure 3-7 Domain Management	46
Figure 3-8 Manager and Service Adapters	47
Figure 3-9 Registry and WS-Context	48
Figure 3-10 System Health Check (SHC) Initialization	52
Figure 3-11 Adding Service Adapter	53
Figure 3-12 System Health Check (SHC) Maintaining System State	54
Figure 3-13 Class diagram of classification scheme in SCGMMS	56
Figure 3-14 SCGMMS sensor filtering mechanism in a distributed architecture	57
Figure 3-15 Event flow when starting a sensor grid domain	60
Figure 3-16 Starting BootstrapService of a Domain	61
Figure 3-17 Normal Health Check Sequence (Stage 1)	62
Figure 3-18 Normal Health Check Sequence (Stage 2)	63
Figure 3-19 Registered Service Adapter (RSA) Health Check Sequence	64
Figure 3-20 Message flow of service adapter discovery in a sensor grid	65
Figure 3-21 Deploying a GPS Sensor	67
Figure 3-22 Disconnecting a sensor by using the Grid Builder management console	68
Figure 3-23 Overall Architecture of Sensor Grid and related Modules	70
Figure 3-24 SG System Management	73
Figure 3-25 Class Diagram of SG, Sensor and Application Client	74

Figure 3-26 Message flow between a Sensor Grid (SG), applications and sensors	78
Figure 3-27 A Sensor Grid startup sequence	79
Figure 3-28 Message flow when depolying a sensor through the Grid Builder	80
Figure 3-29 Sensor Grid message flow during periodic sensor filtering	81
Figure 3-30 Message flow when an application joins a sensor grid	82
Figure 3-31 Message flow from deployed sensors to applications in a sensor grid	83
Figure 3-32 Message flow from a sensor grid to a subscribing application	84
Figure 3-33 Message flow of filter setup in a sensor grid	84
Figure 3-34 Message flow of control messages from applications to sensors in a sensor grid	85
Figure 3-35 Message flow when disconnecting a deployed sensor from a sensor grid....	86
Figure 3-36 SCGMMS Application Programming Interface.....	88
Figure 3-37 A high-level architecture of the Sensor Service Abstraction Layer (SSAL)	89
Figure 3-38 A detailed SSAL architecture for general sensor services	91
Figure 3-39 A detailed SSAL architecture for computation as a sensor service	93
Figure 4-1 UDOP Architecture	96
Figure 4-2 Role of SCGMMS	97
Figure 4-3 Distributed Architecture for Data Access	99
Figure 4-4 A sample GUI-based sensor selection filter.....	101
Figure 4-5 UDOP Management	102
Figure 4-6 UDOP Service Architecture	104
Figure 4-7 System view of UDOP Template management	105
Figure 4-8 Dataflow of saving a UDOP template to UDOP service	105
Figure 4-9 Dataflow of retrieving a UDOP template from UDOP service.....	106
Figure 5-1 A map sharedlet shows live GPS streams of sensor locations from all over the world	108
Figure 5-2 A map sharedlet shows the live GPS stream from Supercomputing 2007 (SC07) in Reno, Nevada	109
Figure 5-3 A map sharedlet shows the live GPS stream from San Francisco, California	109
Figure 5-4 A map sharedlet shows the live stream from Bloomington, Indiana	110
Figure 5-5 A map sharedlet shows the live GPS stream from the Hong Kong International Airport.....	110
Figure 5-6 SCGMMS Multi-location, multi-robot, multi-sensor demonstration scenario	112
Figure 5-7 GPS and video sensor streams from Irvine and Bloomington	113
Figure 5-8 GPS and video sensor streams from Irvine and Hong Kong	114
Figure 5-9 A UDOP dynamically created from remote sensor streams	115
Figure 5-10 A video, Wii Remote and motion edge detection operational picture	116
Figure 5-11 A UDOP for situational awareness around the Irvine demonstration site ..	117
Figure 5-12 Live alert of disconnected GPS sensor.....	118
Figure 5-13 Disconnected sensors in Irvine.....	118
Figure 5-14 A collaborative UDOP user-interface	119
Figure 5-15 A UDOP composed by selecting from an extensible list of sensor streams	120
Figure 5-16 A Video Sharedlet shares four live feeds from four different cities	121

Figure 5-17 A Geo-spatial sharedlet integrating real-time GPS sensor streams	122
Figure 5-18 A UDOP shows and shares RFID signal strength of ten active RFID tags	123
Figure 5-19 A UDOP shows the geo-spatial locations of deployed robots and an observer	123
Figure 5-20 Transformation of live video into live edge-detected video	124
Figure 5-21 Aggregated view of an NXT Robot	125
Figure 5-22 A UDOP shows and shares the eight real-time sensor streams carried by 2 robots with their respective geo-spatial information	126
Figure 5-23 A UDOP shows and shares 4 sensor streams delivered by a Tribot robot..	126
Figure 5-24 A UDOP shows and shares 4 sensors streams delivered by a Humanoid robot	127
Figure 5-25 A UDOP shows and shares RFID signal strength of ten active RFID tags	127
Figure 9-1 Sensor Service Abstraction Layer	149
Figure 9-2 Screenshot of Grid Builder deployment.....	162
Figure 9-3 Screenshot of Sensor Client Program (SCP) extration	162
Figure 9-4 Overview of Application API	164
Figure 9-5 GB Domain Management	185
Figure 9-6 GB Package	187
Figure 9-7 GB Management Console	188
Figure 9-8 Step by step ensor deployment overview.....	193
Figure 9-9 User-interface to start the Bootstrap Service in the ROOT domain.....	215
Figure 9-10 A Bootstrap Console showing statuses of a sensor grid sub-domains	216
Figure 9-11 An initial view of a Grid Builder Management Console	217
Figure 9-12 Overview of Sensor Sharedlet.....	240
Figure 9-13 Sensors on SG sensor hierarchy highlighted in different colors	241
Figure 9-14 Grouping sensors into hierarchies	242
Figure 9-15 Dragging a Lego Mindstorm NXT Humanoid Robot to the top right panel	243
Figure 9-16 Visualization of NXT Humanoid Robot sensor streams.....	244
Figure 9-17 A panel displaying data from GPS device is expanded	245
Figure 9-18 Control panel for NXT Humanoid Robot	248
Figure 9-19 Control panel for XNT Tribot Robot	248
Figure 9-20 VED with Edge Detection Control	248
Figure 9-21 VED with Region Finding Control	249
Figure 9-22 Sensor List before Filtering.....	251
Figure 9-23 Sensor List after Filtering.....	251
Figure 9-24 Setting the Source of Video Service	252
Figure 9-25 Source of VED has been set.....	253
Figure 9-26 Panel for setting mode of operating picture	254
Figure 9-27 View of meeting host in Strictly Synchronous Mode	254
Figure 9-28 View of meeting participant in Strictly Synchronous Mode.....	254
Figure 9-29 View of meeting host in Loosely Synchronous mode	255
Figure 9-30 View of meeting participant in Loosely Synchronous mode	255
Figure 9-31 The Geo-Spatial Sharedlet	256
Figure 9-32 Information of a sensor	257
Figure 9-33 View of meeting participants	257

List of Tables

Table 3-1 Fields of Sensor Property	55
Table 5-1 A sample sensor types and attributes table used in Supercomputing 2007 demonstration.....	107
Table 9-1 A sensor property object.....	150
Table 9-2 A sensor adapter object.....	156
Table 9-3 A sensor grid resource interface	167
Table 9-4 A sensor policy interface	168
Table 9-5 A sensor property interface	169

Table of Contents

NOTICE AND SIGNATURE PAGE	II
REPORT DOCUMENTATION PAGE.....	III
LIST OF FIGURES.....	IV
LIST OF TABLES.....	VII
TABLE OF CONTENTS	VII
1 SUMMARY	1
2 GRID OF GRIDS MIDDLEWARE FOR NET-CENTRIC OPERATIONS.....	3
2.1 INTRODUCTION.....	3
2.2 HYBRID SHARED DISPLAY (HSD) COLLABORATIVE SERVICE.....	6
2.2.1 <i>Lossless Compression</i>	6
2.2.1.1 Entropy Coding.....	6
2.2.1.2 Huffman Coding	6
2.2.1.3 Shannon-Fano	7
2.2.1.4 Dictionary-based Coding.....	7
2.2.1.5 LZ77	7
2.2.1.6 LZ78	7
2.2.1.7 LWZ.....	7
2.2.1.8 Lossless Predictive Coding	7
2.2.2 <i>Lossy Compression</i>	8
2.2.2.1 Lossy Predictive Coding	8
2.2.2.2 Transform Coding.....	8
2.2.2.3 Wavelet Coding.....	8
2.2.3 <i>Implementation Choices</i>	8
2.2.4 <i>Initial HSD Prototype</i>	8
2.2.5 <i>Integrated Hybrid Shared Display in Impromptu Collaboration</i>	11
2.3 NET-CENTRIC COLLABORATION GRID BUILDER TOOL (NCCGBT).....	12
2.3.1 <i>Design Requirements</i>	12
2.3.2 <i>Template Grids</i>	13
2.3.3 <i>Challenging Issues in Design of NCCGBT</i>	14
2.3.4 <i>Interface Design for Net-Centric Collaboration Grid Builder Tool</i>	15
2.4 NCCGBT WITH EARTHQUAKE GRID AND SERVOGRID SUPPORT.....	16
2.4.1 <i>Refined Grid Builder Design and Initial Earthquake Grid Template</i>	16
2.4.2 <i>Support for ServoGrid/QuakeSim and Earthquake Grid Workflow</i>	21
2.5 GRID SERVICES MANAGEMENT ARCHITECTURE AND SYSTEM.....	25
2.6 SUMMARY – GENERALIZING AND PROTOTYPING EXTENDED GRID OF GRIDS TECHNOLOGY	30
2.6.1 <i>Problem Statement</i>	30
2.6.2 <i>Challenges</i>	31

2.6.3	General Goals.....	31
2.6.4	Research Objectives.....	31
2.6.5	Research Methodology.....	31
2.6.6	Research Approach.....	32
2.6.7	Research Tasks.....	32
2.6.8	Part 1 Implementation Status.....	32
2.6.9	CTS 2007 Demonstration.....	33
2.6.10	The Implication of The Demonstration.....	34
3	SENSOR-CENTRIC GRID OF GRIDS	35
3.1	PROJECT GOAL.....	35
3.2	SENSOR-CENTRIC GRID MIDDLEWARE MANAGEMENT SYSTEM ARCHITECTURE.....	37
3.2.1	Grid Builder (GB).....	38
3.2.2	Sensor Grid (SG).....	38
3.2.2.1	Sensor/Sensor Grid flow	38
3.2.2.2	Application/Sensor Grid flow	38
3.2.2.3	Grid Builder/Sensor Grid flow.....	39
3.2.2.4	Application/Sensor flow.....	39
3.2.3	Sensor-Centric Grid Middleware Management System (SCGMMS) API.....	39
3.2.4	Sensor.....	39
3.2.4.1	Sensor Client Program.....	39
3.2.4.2	Computational Service	40
3.2.4.3	Supported sensors	40
3.2.5	Sensor Service Abstraction Layer (SSAL).....	41
3.3	GRID BUILDER	42
3.3.1	An Overview of the Grid Builder Architecture.....	42
3.3.2	Significant Classes	44
3.3.2.1	Class Diagram	44
3.3.2.2	Class Description	49
3.3.3	Important Features	52
3.3.3.1	System Health Check.....	52
3.3.3.2	Classification Scheme.....	55
3.3.3.3	Filtering Mechanism.....	57
3.3.4	Detailed Description.....	60
3.3.4.1	Starting a Domain	60
3.3.4.2	Starting BootstrapService of a Domain	61
3.3.4.3	Normal Health Check Sequence (Stage 1).....	62
3.3.4.4	Normal Health Check Sequence (Stage 2).....	63
3.3.4.5	Registered Service Adapter Health Check Sequence.....	64
3.3.4.6	Service Adapter Discovery.....	65
3.3.5	Deploying and Disconnecting sensors.....	67
3.3.5.1	Deploying a GPS Sensor	67
3.3.5.2	Disconnecting a Sensor	68
3.4	SENSOR GRID	70
3.4.1	Overall Architecture of Sensor Grid and Related Modules	70
3.4.1.1	Message Brokering	71
3.4.1.2	Application Management	73
3.4.2	Significant Classes	74
3.4.2.1	Class Diagram	74
3.4.2.2	Class Description	75
3.4.3	Important Features	77
3.4.3.1	NB Data Flow and Topic Management.....	77
3.4.4	Detailed Description.....	79
3.4.4.1	Sensor Grid Startup.....	79
3.4.4.2	Deploying a Sensor.....	80
3.4.4.3	Periodic Filtering	81
3.4.4.4	Application Client Joining A Sensor Grid (SG)	82
3.4.4.5	Sensor Publishing Data.....	83
3.4.4.6	Subscribing Sensor Data	83

3.4.4.7 Setting a Filter	84
3.4.4.8 Sending Control to a Sensor.....	85
3.4.4.9 Disconnecting a Sensor	86
3.5 SCGMMS APPLICATION PROGRAM INTERFACE (API)	87
SENSOR SERVICE ABSTRACTION LAYER (SSAL)	89
3.5.1 Overall Sensor Service Abstraction Layer Architecture	89
3.5.2 SSAL Architecture for General Sensor Services	91
3.5.2.1 Sensor Deployment.....	92
3.5.2.2 Data Publishing.....	92
3.5.2.3 Performing Actions on Sensor Client Program.....	92
3.5.3 SSAL Architecture for Computation as a Sensor Service.....	93
3.5.3.1 Sensor Deployment.....	94
3.5.3.2 Subscribe Sensor Data	94
4 ADVANCED USER-DEFINED OPERATIONAL PICTURES	95
4.1 UDOP OVERVIEW.....	95
4.1.1 Definitions.....	95
4.1.2 Why UDOP is Needed.....	95
4.1.3 UDOP Architecture.....	96
4.1.3.1 Sensor Layer.....	96
4.1.3.2 Meta Data Layer	96
4.1.3.3 Information Management Layer	96
4.1.3.4 Application Layer	97
4.2 THE ROLE OF SCGMMS IN UDOP	97
4.2.1 How SCGMMS supports UDOP Development.....	98
4.2.1.1 Data Access	98
4.2.1.2 Data Selection and Filtering.....	100
4.2.1.3 Visualization and Presentation.....	101
4.2.1.4 UDOP Management.....	101
4.3 UDOP SERVICE	102
4.3.1 Overview	103
4.3.1.1 UDOP Template.....	103
4.3.2 Architecture.....	103
4.3.3 Detailed Description.....	104
4.3.3.1 Saving a UDOP Template	105
4.3.3.2 Retrieving a list of UDOP Templates	106
4.3.3.3 Opening a UDOP Template	106
5 AN ADVANCED TECHNOLOGY DEMONSTRATIONS OF SCGMMS	107
5.1 DEMONSTRATIONS OF SCGMMS	107
5.1.1 Demonstration of SCGMMS in Supercomputing 2007	107
5.1.2 Demonstration of SCGMMS in CTS 2008.....	111
5.2 A SAMPLE ILLUSTRATION OF USING SCGMMS API FOR UDOP APPLICATIONS.....	119
6 CONCLUSIONS	128
7 RECOMMENDATIONS.....	130
8 REFERENCES.....	133
9 APPENDICE	149
APPENDIX A - USER GUIDE FOR SENSOR DEVELOPERS	149
APPENDIX B - USER GUIDE FOR SENSOR-CENTRIC APPLICATION DEVELOPERS	164
APPENDIX C - USER GUIDE FOR SYSTEM ADMINISTRATOR	176
APPENDIX D - USER GUIDE FOR SENSOR ADMINISTRATOR	184
APPENDIX E - USER GUIDE FOR SCGMMS APPLICATION USER	240
APPENDIX F - RFID POSITIONING (LOCALIZATION)	270
LISTS OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS	273

1 Summary

Problem Statement

Information and communication have played increasingly critical roles in our nation's security. It is a mammoth task with many technical challenges to transform a globally system-centric environment to a net-centric one that offers a single secure information fabric providing end-to-end capabilities to all warfighting, national security and support users; joint, high-capacity netted operations fused with weapon systems; strategic, operational, tactical and base/post/camp/station levels with a common operating picture; and making tactical and functional fusion a reality.

The emergence of the GiG and Net-Centric systems will bring about new opportunities for seamless connectivity for the 21st century warfighter. However, as the worldwide network of the Department of Defense (DoD), the GiG is not one global seamless construct. It has many pieces, interconnecting with one another. Each of these often has different stakeholders with different missions. The services each have their own piece of the GiG, with its own name and unique vision of network-centric operations. Many operations have been done independently, in some cases by different chains of command. The GiG is inevitably a Grid of Grids.

Sensor and collaboration technology play critical roles in supporting war fighters and military personnel as they engage in operations that could be high stress and life threatening. While the Global Information Grid (GiG) will provide the global connectivity in the net-centric environment for the information to get to where it's needed, however the information will still need to be gathered, filtered, processed and converted to knowledge to empower our warfighter to make the right decisions and succeed. Moreover, it will take collaborative teamwork on the part of a collection of geographically distributed domain experts to process these vast amounts of information to support effective decision making in this futuristic NCOW-GiG environment.

Results

A general-purpose grid building and management system with a particular focus on sensor-centric grid of grids, called the Sensor-Centric Grid Middleware Management System, has been designed and implemented. The Grid Builder provides an intuitive interface to set resource or sensor policies as well as easy deployment and management of resources across global networks. SCGMMS is deployable as a distributed prototype for effective and efficient support of User-Defined Operating Picture (UDOP) and Common Operating Picture (COP) applications. Two key design objectives of SCGMMS are its support for easy and simple system integration interface for any third party application and sensor developers. During the course of SBIR, there was substantial technology evolution in especially mainstream commercial Grid applications. These evolved from (Globus) Grids to clouds allowing enterprise data centers of 100x current scale. This change impact Grid components supporting background data processing and simulation as these need not be distributed. However Sensors and their real time interpretation are naturally distributed and need traditional Grid systems. Thus SCGMMS is not directly impacted by these shifts and can take direct advantage of cloud

systems. Further experience has simplified protocols and deprecated use of some complex Web Service technologies but we had anticipated this in using light weight service architectures. In this modern terminology, SCGMMS develops Sensor as a Service that integrated with software as a service and computing/storage as a service builds a complete grid when implemented on the GiG – Infrastructure as a service.

Conclusions

The message-based SCGMMS prototype in association with the sophisticated UDOP/COP capable Impromptu collaborative sensor sharedlet application demonstrates the power of the system and its robust, extensible architecture and implementation for providing critical situational awareness for warfighters and human decision-makers in the loop. It offers a very easy to implement application development interface for integrating legacy or new third-party applications with a sensor grid to add crucial grid situational awareness capability. It also provides a concise layered sensor service abstraction for easy integration and deployment of new sensors as a grid resource to enhance grid situational awareness.

2 Grid of Grids Middleware for Net-Centric Operations

In Phase I Anabas pioneered the development of the Grid of Grids architecture, and has proven and demonstrated the feasibility of the concept by integrating and making interoperable of separately developed sensor, data-mining, GIS and archiving grids using publish-subscribe based mediation service. Anabas developed and demonstrated in Phase I effort not only an architecture but also developed several unique capabilities and innovations in software for Grid of Grids that are needed in order to satisfy critical Net-Centric Operational Warfare objectives. These critical capabilities include, among others, a network-based global clock for service-oriented architecture (SOA) without which it will very difficult if not impossible to support the important time-criticality requirement in NCOW applications; novel methods to support uniform treatment of grid and Web Service thus enabling a powerful composition model to support plug-and-play integration of legacy systems and next-generation transformational DoD grids built with different profiles; and novel methods and apparatus to support desired levels of QoS within different Communities of Interest (COIs), and the scaling of Web Services especially in areas of QoS including reliable messaging. These unique and significant innovations and capabilities proven in Phase I are crucial and necessary technology elements in order to fulfill both global interoperability and COI-level specialization/customization for DoD's NCOW vision.

Based on the successful feasibility study and the developed Grid of Grids software system demonstration Anabas proposed for Phase II effort to develop a suite of Net-Centric Collaboration Grid Middleware and Collaboration Community Grid Builder and User-Defined Operational Picture tools that uses Grid of Grids architecture as a base to prototype a complete and enhanced Net-Centric Enterprise Services (NCES) Core Enterprise Services. As anticipated, the Phase II research and development effort led to the development of an advanced technology demonstration of two standards-compliant and functionally complete prototypes: A particular Sensor-Centric Collaboration Grid Middleware Management System (SCGMMS) with User-Defined Operational Picture capability (UDOP) and a Community Collaboration Grid Building Tool generically called the Grid Builder (GB).

2.1 Introduction

Information and communication have played increasingly critical roles in our nation's security. It is a mammoth task with many technical challenges to transform a globally system-centric environment to a net-centric one that offers a single secure information fabric providing end-to-end capabilities to all warfighting, national security and support users; joint, high-capacity netted operations fused with weapon systems; strategic, operational, tactical and base/post/camp/station levels with a common operating picture; and making tactical and functional fusion a reality.

The emergence of the Global Information Grid (GiG) within the next few years will bring about new opportunities for seamless connectivity for the 21st century warfighter. The GiG will encompass a sophisticated interconnection of hardware pipes, satellite links and tactical communication links providing a vast amount of information directly to the

warfighter. Furthermore, Network Centric Enterprise Services will be a key core capability for implementing the GiG.

However, as the worldwide network of the Department of Defense (DoD), the GiG is not one global seamless construct. It has many pieces, interconnecting with one another. Each of these often has different stakeholders with different missions. The services each have their own piece of the GiG, with its own name and unique vision of network-centric operations. Many operations have been done independently, in some cases by different chains of command.

In reality, the GiG is an umbrella. For instances, the Air Force has C2 Constellation, Navy has FORCENet and Army has LandWarNet. Each service has to do plan, management and operate in a way that makes sense to their missions and satisfies their unique requirements. Each of them has a shared challenge of providing vertically seamless, secure and interconnected environment for users from the home stations all the way to the warfighters within their respective services, and horizontally to other joint community and coalition forces.

To achieve the GiG and NCOW vision requires among other things several operational tasks including network management, enterprise services management, information staging and dissemination management be done across the network using common tactics, techniques, and procedures. These activities must be synchronized and integrated in order to be able to provide the joint forces the ability for better situational awareness, ad-hoc synchronization and speed of command and action.

The biggest challenge is the necessary balance of the competing demands for standardization, customization and modernization. Standardization is an enabler for interoperability. However, over-standardization could jeopardize needed flexibilities particularly for warfighters in theaters of operations. Customization acknowledges the different nature of the services and needs of personnel at different levels within the services and across joint forces and alliances. Customization generally makes systems and capabilities difficult to interoperate with other systems at the joint level. Modernization is a measure of staying “current” which suggests an architecture that could accommodate and seamlessly integrate the latest relevant commercial IT product and service offerings that are moving fast towards solving many of the hard information management problems of interest to DoD in a generic fashion.

The initial Grid of Grids technology that Anabas successfully demonstrated its feasibility of in Phase I is a key enabler to address the challenge of balancing interoperability, customization, and modernization. The Grid of Grids could support flexible interoperability and customization at all levels, and facilitates seamless integration of relevant COTS offerings. The modular architecture supports related Grids with multiple critical infrastructures and multiple natural and man-made triggers.

Anabas’ novel idea and method of treating services and grids uniformly using the Grids of Grids concept is critical to the success of the project. It allows a powerful composition model that can link legacy systems and grids built with different profiles. As demonstrated in Phase I the Grid of Grids concepts for NCOW-type of applications are feasible but need significant development, refinement and optimization to bring them to a commercially usable state.

The Phase 2 work built upon the initial result of Phase I work, and defined a technical program that laid a necessary solid foundation for successful commercialization in follow on efforts. The information formulated during Phase II will allow transfer of technology to other Air Force and military components and to the commercial market.

Our Phase 2 research and development was divided in two parts. In Part 1, general Grid of Grids issues and the development of a Collaboration Grid Middleware and Net-Centric Collaboration Grid Builder Tool were studied and system components prototyped and experimented. Leveraging on the results of Part 1, a Sensor-Centric Grid of Grids Middleware Management System (SCGMMS) and Grid Builder tool with a particular focus on integrating distributed sensors, robots, GIS, modeling and simulation tools for earthquake crisis management, and computational software as a service was designed, developed and demonstrated. The adoption and extension of Part 1 results for Sensor-Centric Grid of Grids Middleware Management System (SCGMMS) and the SCGMMS Grid Builder tool are discussed in Section 3.

A preview of integrating QuakeSim, a sample Earthquake Grid of Grids application for earth science modeling and simulation, with SCGMMS is illustrated in Figure 2-1:

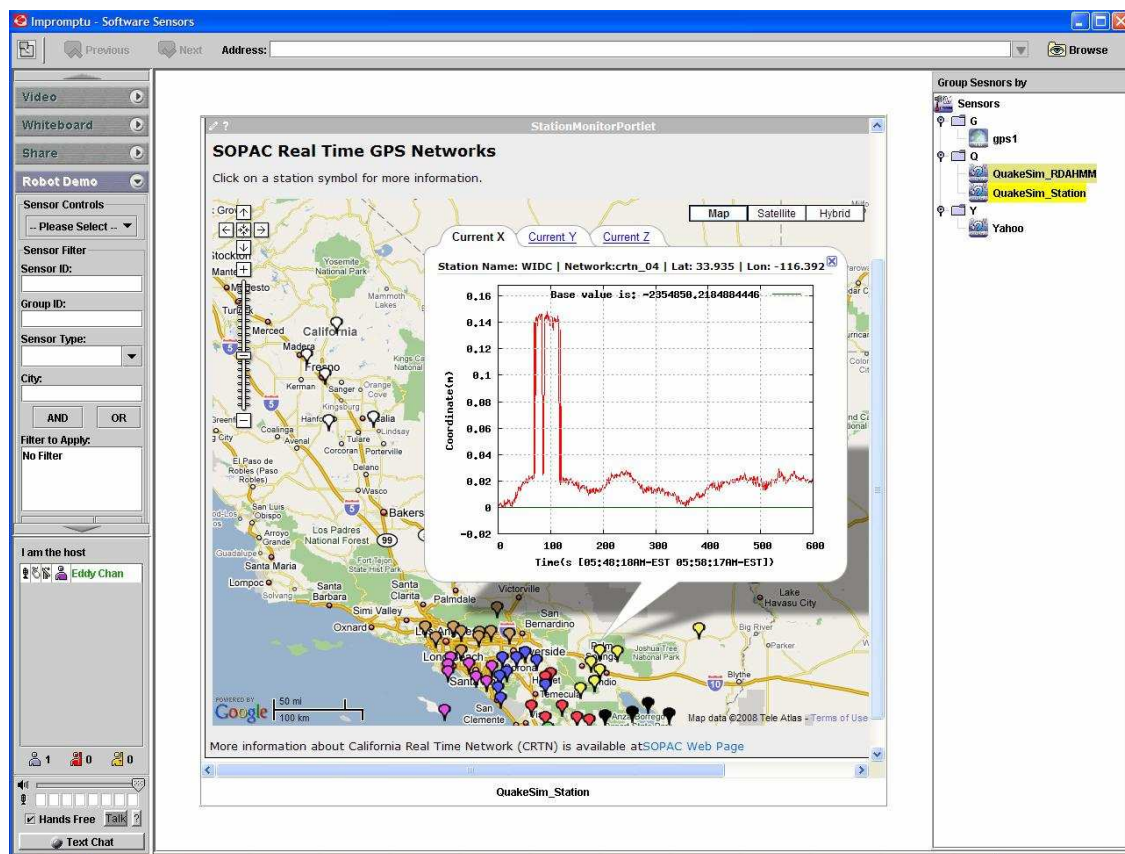


Figure 2-1 Integrating modeling and simulation systems with real-time and archived GPS sensor streams into Sensor-Centric Grid of Grids

The rest of Section 2 is organized to reflect the R&D activities that were undertaken to generalize and prototype Grid of Grids technology for future commercialization. In Section 2.2, we discuss a high-performance, core collaboration service for the collaboration grid middleware called hybrid shared display. In Section 2.3, the initial design and prototyping of our Net-Centric Collaboration Grid Builder Tool (GB) is discussed. Grid Builder support for EarthquakeGrid, ServoGrid, workflow editor and grid resource viewer design are covered in Section 2.4. In Section 2.5, we discuss the design of user-assisted interface for Grid Builder and the initial design of a grid service management architecture. We summarize the generalization and prototyping Grid of Grids technology in Section 2.6 as a foundation for Part 2 – the Sensor-Centric Grid of Grids Middleware Management System and SGCMMMS Grid Builder.

2.2 Hybrid Shared Display (HSD) Collaborative Service

Anabas had an invention called HSD – Hybrid Shared Display - that could significantly improve a key collaborative service based on XML messaging for interactive, multi-point, multimedia sharing. HSD was designed to address performance and bandwidth utilization issues in most synchronous, interactive sharing of large volume of streaming data in a net-centric environment.

For HSD to work, the algorithm needs to determine or intelligently classify regions on the framework into fast or slow changing groups. Region finding, even when limited to parallel rectangular areas, is a typical ill-posed computer vision problem. Experience with such problems suggests that we try various complementary approaches in parallel and we make the final decision by fusing all available information.

We include here a brief non-exhaustive overview of various image compression algorithms relevant for the discussion in this report.

2.2.1 Lossless Compression

There are various compression algorithms that are lossless. We reviewed several popular ones here for background information.

2.2.1.1 Entropy Coding

Entropy Coding is based on statistical / information theory considerations and it tries to find the optimal codes that saturate Shannon theory bounds. The first such algorithm was developed in '50s by Shannon and Fano (called Shannon-Fano) and soon after slightly improved by Huffman in '52 – now this class of algorithms is known under the name of Huffman codes.

2.2.1.2 Huffman Coding

Huffman algorithm assigns codes to symbols in such a way that more frequent symbols have shorter codes and each code can be uniquely decoded. The algorithm is based on Huffman tree which is a binary tree with symbols as leaves, constructed based on the symbol frequency values. The tree construction for n symbol alphabet starts with a set of n one-node trees and terminates with a single tree including all symbols. At each step, two trees with lowest frequency values are removed from the tree set, merged as a single

tree with the root frequency given as a sum of tree frequencies, and appended back to the tree set. This process continues until n trees are reduced to one single tree with all symbols as leaves. Codes are built as binary tree addresses of the corresponding symbols (0 to move left, 1 to move right). Decoding amounts to traversing the tree until the leave indicated by the address is reached.

2.2.1.3 Shannon-Fano

Shannon-Fano is similar to Huffman but it constructs the tree in the top-down fashion using frequency weighted recursive bisection.

2.2.1.4 Dictionary-based Coding

Dictionary-based codecs replace (longer) patterns (phrases, words, groups of bytes) by (shorter) references to previous occurrences of these patterns in the source. Hence, the source acts itself as a dictionary / lookup table. Several popular compression algorithms such as zip or gif belong to this category.

2.2.1.5 LZ77

The first dictionary-based codec was constructed in 1977 by Lempel and Ziv, and is now known as LZ77 algorithm. LZ77 encoder uses two sliding windows: a) search buffer that contains a portion of the recently encoded sequence; and b) look-ahead buffer that contains the next portion of the sequence to be encoded. Encoder moves the pointer through the search buffer, selects the longest match, encodes it as <offset, length, next> tuple where next is the first symbol in the look-ahead buffer after the sequence. If no match was found, the symbol s is encoded as <0,0,s>. Decoding is based on lookup that uses the already decoded sequence as a dictionary. LZ77 has known problems with window size uncertainty and codec efficiency and it seems to be now only of historical relevance as the first step towards a family of more powerful algorithms such as LZ78 and LZW.

2.2.1.6 LZ78

Follow-on algorithm by the same authors that does not use sliding window for the search buffer but instead it builds on-the-fly a dictionary of phrases encountered in the scanning process. There are no size limits for the search buffer and the codec is more efficient (no need to specify sequence length).

2.2.1.7 LWZ

Most recent instance in the LZ series by Terry Welch '84. The dictionary starts with 256 single character entries. The encoder keeps adding new string entries while reading symbols. Decoder reconstructs the dictionary while reading the codes and uses it as lookup for decoding. A version of LZW is patented and used in GIF.

2.2.1.8 Lossless Predictive Coding

Predictive coding includes predictor that estimates the code for the next symbol based on the code for the current symbol and corrector that calculates an error of such estimate. The goal is to represent the error as a shorter code than the symbol itself. A simple

example is to predict the value of the next pixel to be the same as the current pixel in the scan line, or as an average over neighbor pixel values

2.2.2 Lossy Compression

We reviewed three popular lossy compression algorithms here.

2.2.2.1 Lossy Predictive Coding

Lossy Predictive Coding uses a similar predictor/corrector approach as the lossless version but it also includes error quantization that forces errors to be represented as shorter codes but it might introduce information loss.

2.2.2.2 Transform Coding

Transform Coding performs a transform of the original code to another coordinate frame / space where some lossy compression such as predictive coding can be more efficient than in the original frame/space. Typical example is given by Discrete Cosine Transform (DCT) used by JPEG and MPEG. Other examples are H.261 and H.263.

2.2.2.3 Wavelet Coding

Wavelet Coding is similar to Transform Coding but more complex due to additional transformational degrees of freedom such as scaling functions. Wavelet compression follows similar steps as DCT algorithms – it is more computationally intense but also more scalable (can be stopped at any resolution). It is often more efficient for low bit rates as it produces less objectionable (more smooth) artifacts than DCT.

2.2.3 Implementation Choices

We identified four region-finding techniques for developing the XML message-based HSD sharedlet. They are

Region Growing – scan blocks after fast or slow classification and grow regions based on some proximity rules. At the end of the scanning, these boundingrectangles are to be passed to the lossy encoder of choice.

Edge Detection and Grouping – perform Hough Transform in X and Y direction when scanning in X and Y direction.

Method of Moments - Consider a distribution that is 1 for F blocks and 0 for S blocks. Compute first and second moments i.e. average/median and dispersion/variance in both X and Y directions.

Monitoring Windows Event – if the fast changing region is due to streaming video, it may be possible to track the windows event causing the changes and determine a bounding rectangle that way.

2.2.4 Initial HSD Prototype

We constructed a region finding algorithm based on method of moments. The current prototype implementation is free of any parameters. Initial test results are discussed.

The following screendumps illustrate the visualization tool (vis tool) for the moments based region finding algorithm. As always, Host/Presenter Shared Display (SD) is in the bottom right corner, client SD is in the top left corner, and the vis tool is in the bottom left corner.

The vis tool displays in real-time i.e. in parallel with the based SD operation, the effective variance image. The vis screen is updated twice a second and the bounding rect angle is computed for each update. The coordinates of the last 10 rectangles (computed within the last 5 seconds) are displayed in the console window below the variance image.

In Figs 1-3, the last (current) rectangle constructed by the algorithm is also displayed in red on top of the variance image in the vis tool. The following four screendumps (Figs 4-7) display all 10 rectangles, constructed by the algorithm within the last 5 seconds. This way, one can observe the time variance of the video rectangle, constructed using the method of moments. Some initial observations for various video domains are included in the figure captions below.

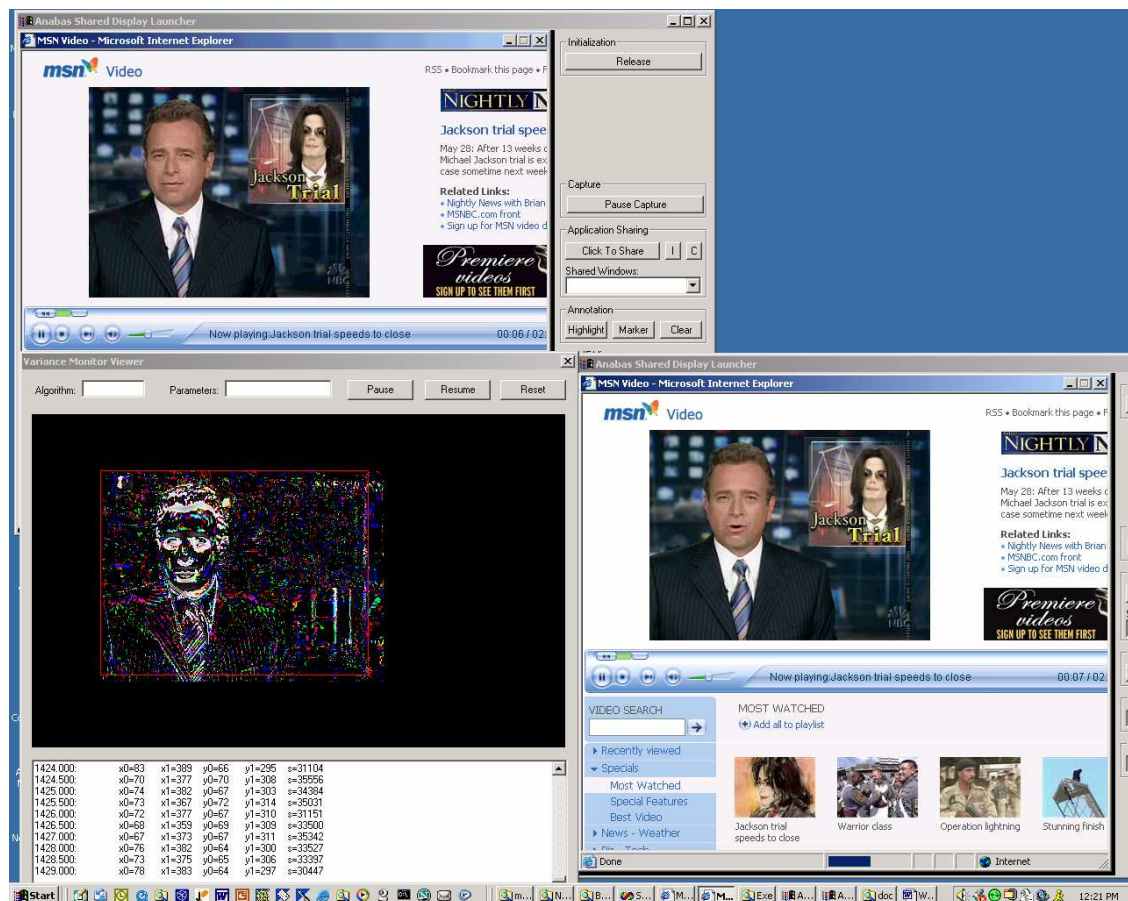


Figure 2-2 A typical variance pattern, observed for a 'talking head' video segments.

The background used in Figure 2-2 is typically static and the leading time variance is generated by small facial changes and body movements. In consequence, the bounding rectangle is off its real video value and often depends on the speaker position (the error is higher if the speaker is not in the center). As seen, we are getting some non-zero background variance pattern scattered over the whole screen.

The effectiveness of the algorithm to detect a fast changing region is illustrated in Figure 2-3 for a video sequence with rapid panning and zooming operations. Because of the drawing pattern, the background noise is reduced – as manifested by large patches of black (motion that is undetected as the pixel intensity is constant). However, rapid zoom generates enough variance and the detected rectangle is again reasonable.

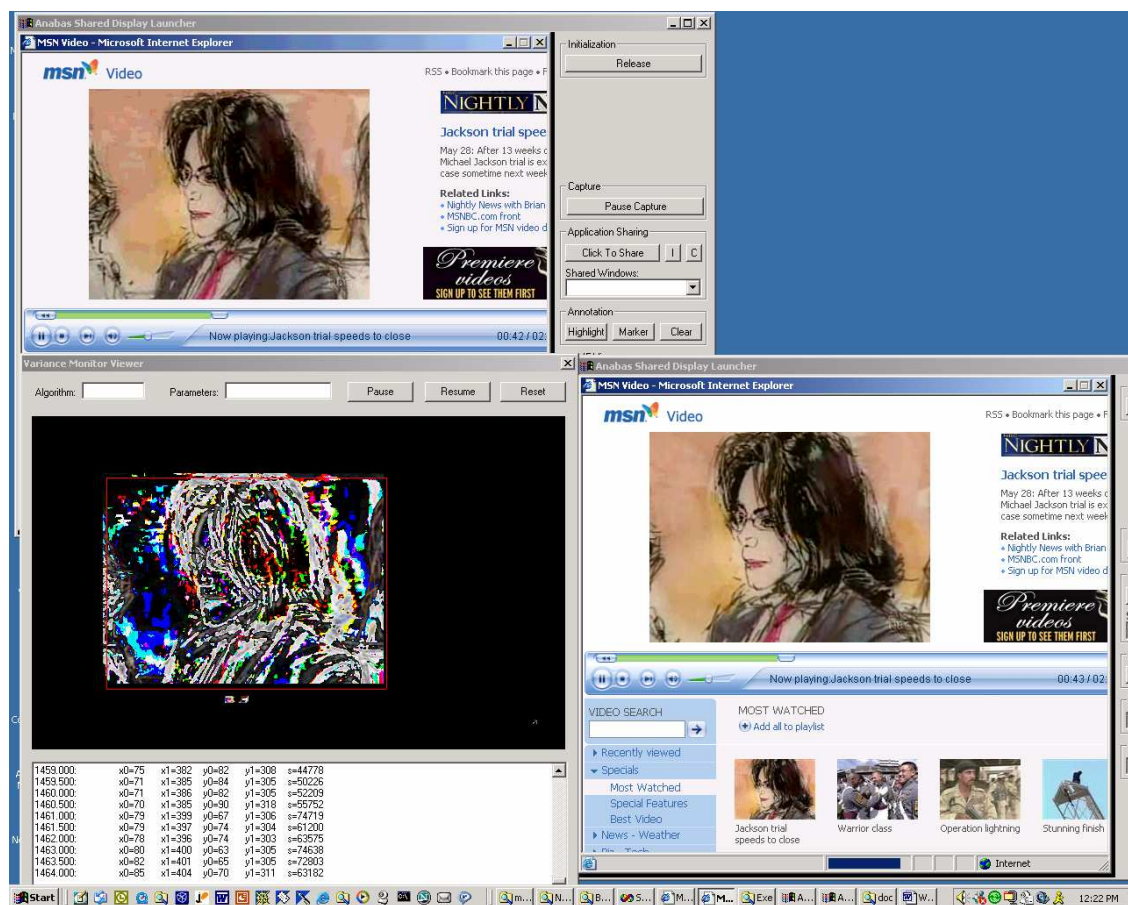


Figure 2-3 Region detection on a typical video zooming and panning sequence.

For comparison purposes, the same algorithm is applied to another video panning sequence illustrated in Figure 2-4. This sequence has enough variance across the whole image and the method of moments generations a better result than in the previous two cases.

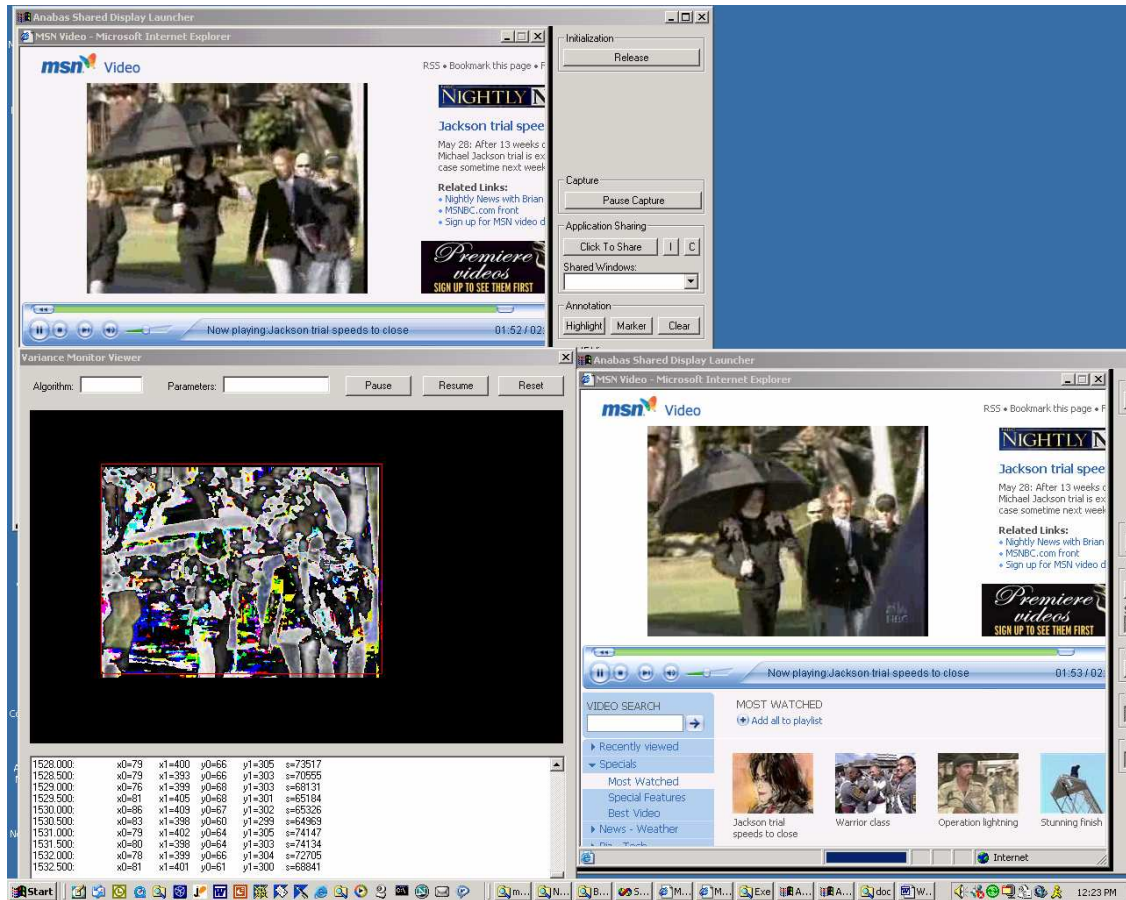


Figure 2-4 Region detection on another typical video panning sequence.

2.2.5 Integrated Hybrid Shared Display in Impromptu Collaboration

We built an HSD sharedlet that integrates lossy and lossless compression for synchronous collaboration of some typical Web pages or applications with video or animation. This type of pages put a lot of stress on the network and performance when lossless codec is applied on them. The HSD sharedlet uses H.261 for its lossy codec. H.261 is a high-performance codec normally used in video conferencing. An example of the fully integrated prototype of HSD sharedlet as one of the real-time collaboration capabilities in an Impromptu client is illustrated in Figure 2-5.

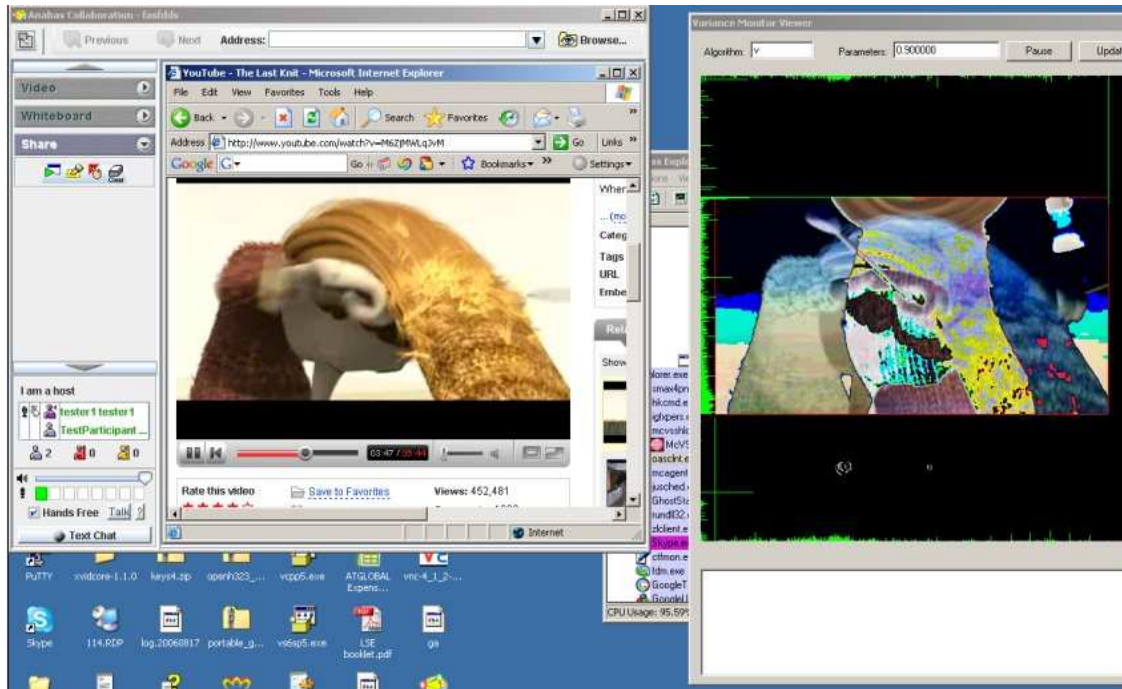


Figure 2-5 Fast changing region detection in an Impromptu client.

The video portion of the shared Web page was coded in H.261. Using normal lossless codec on fast changing region will stress both the computation and network utilization. Using HSD in this case, not only can the fast changing region be shared at video frame rate, the static region which is much smaller than the whole shared Web page can be shared by the higher quality, lossless codec. Network stress is now also under controlled due to the use of more sophisticated and a higher compression ratio algorithm in H.261.

2.3 Net-Centric Collaboration Grid Builder Tool (NCCGBT)

The goal here is to design and implement a tool to facilitate users to build grids from existing libraries of services and grids, including the features below:

- Dynamic resource assignment and management
- Real time requirement
- Extending an existing BPEL workflow engine
- Template Grids
- Customizable portal/dashboard
- Specifying contexts

2.3.1 Design Requirements

- A Grid Builder is used by the administrator to provision resources for the services and link to the workflow. Appropriate portlets are needed for end users to view

available services and resources. The portal should be dynamically customized to current requirements.

- Grid Builder deals with existing capabilities and assembles into an operational grid by assigning grids/services to resources while developing new algorithms or significant new data is not considered.
- Grid Builder has libraries of grids/services and template grids corresponding to certain scenarios. It can instantiate copies of template grids corresponding to existing differently deployed grids.
- Grid Builder can link Grids (in Grid of Grids model) generating mediation infrastructure. It includes a dynamically assembled layout of interoperating portlets and effectively determines the portlets and layouts. Appropriate views will be provided for end users. The mediation infrastructure can include Quality of Service and fault tolerance support.

2.3.2 Template Grids

- The Grid Builder can instantiate template grids taking into account scale of events and currently available resources. Each template grid corresponds to an anticipated scenario. It can copy (with perhaps no changes except to host machines) a Grid from one deployment to another; from an IU earthquake deployment to an Anabaz deployment. It can generate the needed management (fault tolerance, monitoring, and firewall strategies).
- There are different levels of abstraction for template grids. Some well-defined template grids can be quickly instantiated by layman users to handle emergencies. Expert users may choose a more generic template to achieve more flexibility.
- The library of template grids can be organized into different categories, which are indexed by their themes. For example, a category with the theme of disaster rescue may include template grids for earthquakes, hurricanes, etc. Thus, the user can select the correct template grid for the current situation more conveniently.
- There should be alternative template grids so that there are backups in case that the previously selected template grid cannot operate successfully.
- A template grid may be a grid of grids. It can be composed by multiple template grids in a hierarchical structure.
- Possible attributes of a template grid and their values are listed in the following table:

Attributes		Possible Values
Service Type	Application Services	GIS, Sensor, Filter, etc
	System Services	Security, Management, Registry, etc
Operating System		Windows, Linux, Solaris, etc
Server container		IIS, Apache, etc
Context		Firewalls, Running environment, SOAP types, NATs, etc
Database		JDBC, Oracle, MS SQL, etc
Client		Portal, Matrix, Anabas, Dashboard, etc

Figure 2-6 A sample template grid design

2.3.3 Challenging Issues in Design of NCCGBT

NCCGBT required several challenging issues listed below to be addressed

1. Representation of a template grid
2. How to determine attributes that a template grid needs to include?
3. How to represent the workflow in a template grid?
4. How to manipulate a template grid?
5. Customization of UI
6. How to deploy the established grid?
7. How to validate the prototype of Grid Builder?
8. How to nest Grid Builder in Matrix?
9. How to support collaboration in Grid Builder?

Future work should include emerging technologies especially those from the rapidly evolving cloud arena. We recommend building in support for the new open source cloud environment Eucalyptus mimicking Amazon and Google interfaces. One should support MapReduce workflow and the powerful database and file system abstractions supported in clouds.

2.3.4 Interface Design for Net-Centric Collaboration Grid Builder Tool

Eclipse was used as the development environment. Its plug-in architecture allows easier extension for rapid prototyping. We chose to experiment with Web Services Business Process Execution Language for use in NCCGBT. A screenshot of a BPEL designer, a plug-in that will be extended to construct a template grid is shown below.

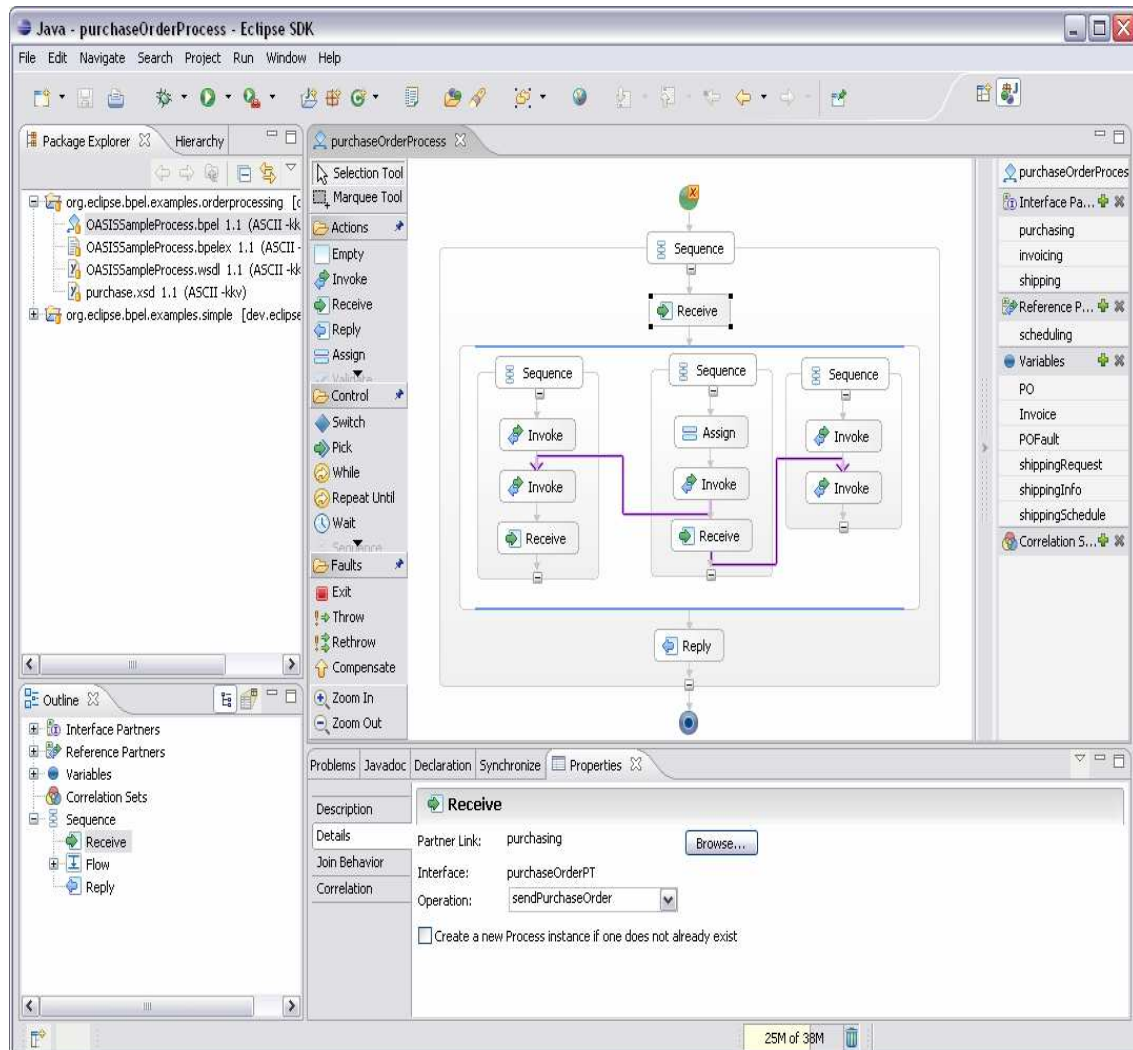


Figure 2-7 An Eclipse-based Grid Builder Tool Layout Design

The architecture for monitoring currently available resources (e.g., machines, service instances, sensors, etc.) and their status is illustrated here:

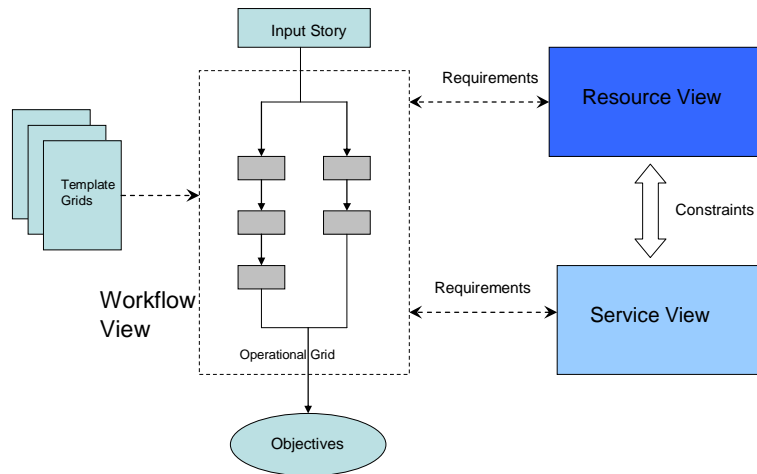


Figure 2-8 Resource management architecture in Grid Builder

To sum up,

1. Grid Builder handles real time situation corresponding to scheduling systems.
2. Grid Builder focuses on resources/services management and provision to facilitate workflow while the execution of workflow is not covered.
3. Use Eclipse BPEL to develop the initial Grid Builder tool that connects to workflow.
4. Predefined Grid templates that specify certain requirements need to be included to customize resource assignments of services and the portal/dashboard.
5. Integrate Grid Builder with Matrix (as sub-grids).
6. Support of adaptive workflow to handle dynamic situations (e.g., failures, changes, etc).

2.4 NCCGBT with Earthquake Grid and ServoGrid Support

2.4.1 Refined Grid Builder Design and Initial Earthquake Grid Template

The Grid Builder tool is being spirally implemented and is being initially experimented with Earthquake Grid. We have

- Extended the EMF model that represents the WS-BPEL 2.0 specification by including additional information to model a template grid. The model specification is described in XML.
- Sample Earthquake Grid template - Below provides an initial example of a template for the Earthquake Grid (the specifications for sub grids such as GIS Grid, Sensor Grid are described in the same way in separate templates):

```
<Grid name="Earthquake"
targetNamespace="http://cgl.com/ws/earthquakegrid"
xmlns="http://schemas.xmlsoap.org/ws/earthquake-process/"
xmlns:lns="http://manufacturing.org/wsd/earthquake">
```

```
<services>
<applications_services> name="GISGrid"
targetNamespace="http://acme.com/ws-bp/gisgrid"
myRole="GISInfo"/>
<application_services> name="SensorGrid"
targetNamespace="http://cgl.com/ws /sensorgrid"
myRole="SensorInfo"/>
<system_services> name="Security"
targetNamespace="http://acme.com/ws-bp/security"
myRole="SSH"/>
<system_services> name="Registry"
targetNamespace="http://acme.com/ws-bp/registry"
myRole="UDDI"/>
<system_services> name="Management"
targetNamespace="http://acme.com/ws-bp/management"
myRole="QoS"/>
</services>
```

```
<variables>
<variable name="OS" Type="Linux"/>
<variable name="Container" Type="tomcat"/>
<variable name="Message" Type="SOAP"/>
<variable name="Database" Type="ODBC"/>
<variable name="Client" Type="Matrix"/>
</variables>
```

```
<contexts>
<firewalls Type="" Status = "on"/>
<nats type="" Status = "enabled"
<message type="SOAP"/>
</contexts>
```

```

<bpel_workflow>
<faultHandlers>
<catch faultName="lns:cannotComplete"
faultVariable="Fault">
<reply partnerLink="administrator"
portType="lns:fault"
operation="sendError"
variable="POFault"
faultName="cannotCompleet"/>
</catch>
</faultHandlers>

<sequence>
<receive partnerLink="GIS"
portType="lns:Filter1"
operation="CheckGISInfo"
variable="PO">
</receive>
<links>
<link name="switch_to_Sensor"/>
</links>
<assign name="Sensor_Info">
</assign>
<invoke partnerLink="switching"
operation = "submitJob"
....
</invoke>
<receive partnerLink="switching"
portType="lns:earthquakeinfo"
operation="send_location_of_earthquakes"
variable="filter">
</receive>
</sequence>

<bpel_workflow>
</Grid>

```

An NCCGBT prototype based on the open source Eclipse BPELDesigner:

- Extends the current BPEL model by adding an element named “Header”, which includes attributes of a template grid such as type, OS, category, resources, etc.
- Registers a *BPELHeaderSerializer* and *BPELHeaderDeserializer* for the *ExtensibilityElement* Head in *BPELExtensionRegistry*.
- Modifies the EMF.ecore model *bpel.ecore* to include the new element and generate necessary model code for it. The new model is viewed as below:

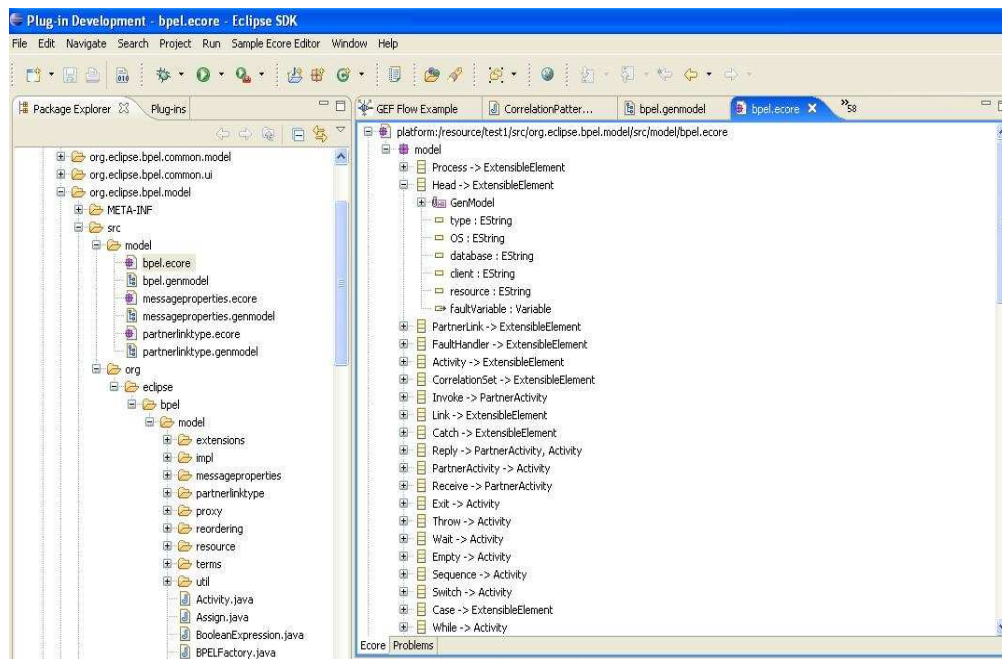


Figure 2-9 An NCCGBT supported EMF model view of BPEL-extended elements

The grid templates may be stored in the registry from the library so that a manager can extract a copy and instantiate it automatically. Resources will be allocated to grid services by the manager. A workflow engine (ActiveBPEL) will be responsible for deploying and managing the workflow defined in the grid. In this way, the reliance on human assistance in establishing grids is greatly reduced.

2.4.2 Support for ServoGrid/QuakeSim and Earthquake Grid Workflow

The Solid Earth Research Virtual Observatory Grid (ServoGrid) is a system with grid services and portals to support earthquake science. The development of ServoGrid/QuakeSim is a collaborative effort among researchers in JPL, UC-Davis, USC, Brown and Indiana University.

“QuakeSim is a project to develop a solid Earth science framework for modeling and understanding of earthquake tectonic processes. The multi-scale nature of earthquakes requires integrating many data types and models to fully simulate and understand the earthquake process” (<http://quakesim.jpl.nasa.org>).

Basically, The QuakeSim (Earthquake Grid) portal includes a number of portlets and services:

- Portlets
 - RDAHMM-portlet (Regularized Deterministic Hidden Markov Model)
 - STFILTER-portlet (time series filter portlet)
 - StationMonitor-portlet
 - Gridsphere
 - RealTimeRDAHMM-portlet
- Execution Services
 - Analyze TseriService
 - AntVisco (GeoFest, etc.)
 - GnuplotService
 - RDAHMMService
 - STFilterService

The updated design of NCCGBT to support ServoGrid/QuakeSim workflow process and its mapping to execution services are illustrated below:

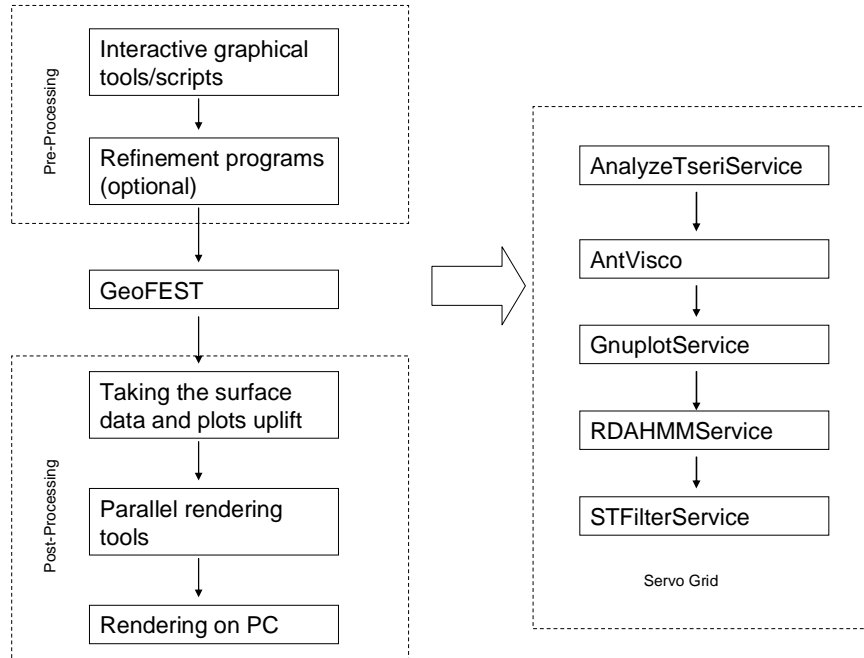


Figure 2-11 An updated design for Grid Builder to support ServoGrid/Quakesim

Also, a screenshot of an enhanced BPEL editor for Grid building in NCCGBT by implementing an extension of the graphical BPEL editor with a new interface is show as below in **Figure 2-12** (there is a new category named “attributes” on the left column to describe features of the template grid):

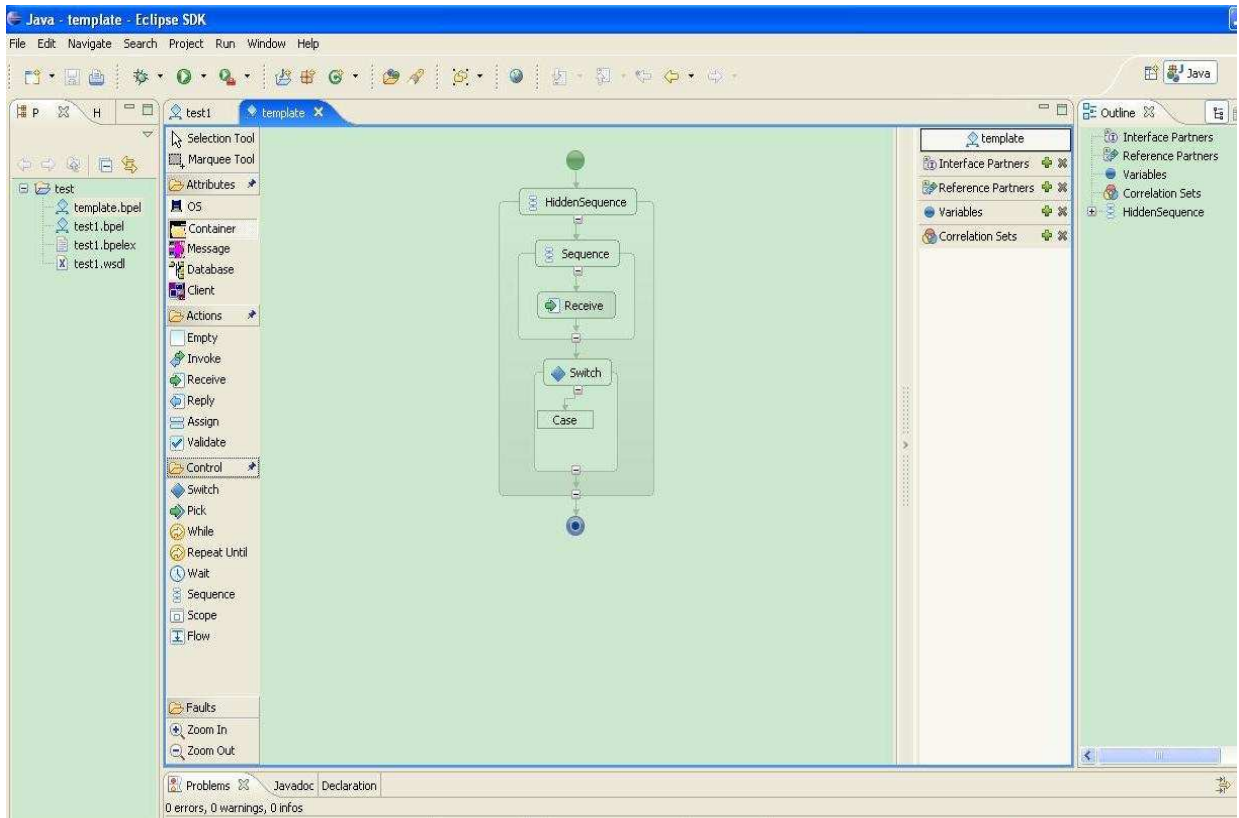


Figure 2-12 An enhanced BPEL editor for Grid Builder

A major modification was instrumented in the class `org.eclipse.bpel.ui.BPELEditor`

```
private void createBPELPaletteEntries(PaletteContainer
palette) {
.....

    PaletteCategory headerCategory = new
PaletteCategory("Attributes");
    headerCategory.add(new
MyPaletteItem("OS", "Operation System",

        provider.getFactoryFor(bpelPackage.getEmpty()), BPELUIP
lugin.getPlugin().getImageDescriptor("obj16/os.gif"), BPELUI
Plugin.getPlugin().getImageDescriptor("obj20/os.png")));
```

```

        headerCategory.add(new
MyPaletteItem("Container", "Service Container",

        provider.getFactoryFor(bpelPackage.getInvoke()), BPELUI
Plugin.getPlugin().getImageDescriptor("obj16/container.gif"
), BPELUIPlugin.getPlugin().getImageDescriptor("obj20/contai
ner.png")));

        headerCategory.add(new
MyPaletteItem("Message", "Message Format",

        provider.getFactoryFor(bpelPackage.getInvoke()), BPELUI
Plugin.getPlugin().getImageDescriptor("obj16/communication.
gif"), BPELUIPlugin.getPlugin().getImageDescriptor("obj20/co
mmunication.png")));

        headerCategory.add(new
MyPaletteItem("Database", "Database",

        provider.getFactoryFor(bpelPackage.getInvoke()), BPELUI
Plugin.getPlugin().getImageDescriptor("obj16/database.gif")
, BPELUIPlugin.getPlugin().getImageDescriptor("obj20/databas
e.png")));

        headerCategory.add(new
MyPaletteItem("Client", "Client Name",

        provider.getFactoryFor(bpelPackage.getInvoke()), BPELUI
Plugin.getPlugin().getImageDescriptor("obj16/client.gif"), B
PELUIPlugin.getPlugin().getImageDescriptor("obj20/client.pn
g")));

        palette.add(headerCategory);

}

```

Another enhancement to NCCGBT is the implementation of an intuitive user-interfacce for viewing available grid resources (Figure 2-13). The purpose of the resource view is to enable users to view current status of available resources so that they can select the right resources to satisfy the requirement of the workflow in an easy to comprehend and efficient manner.

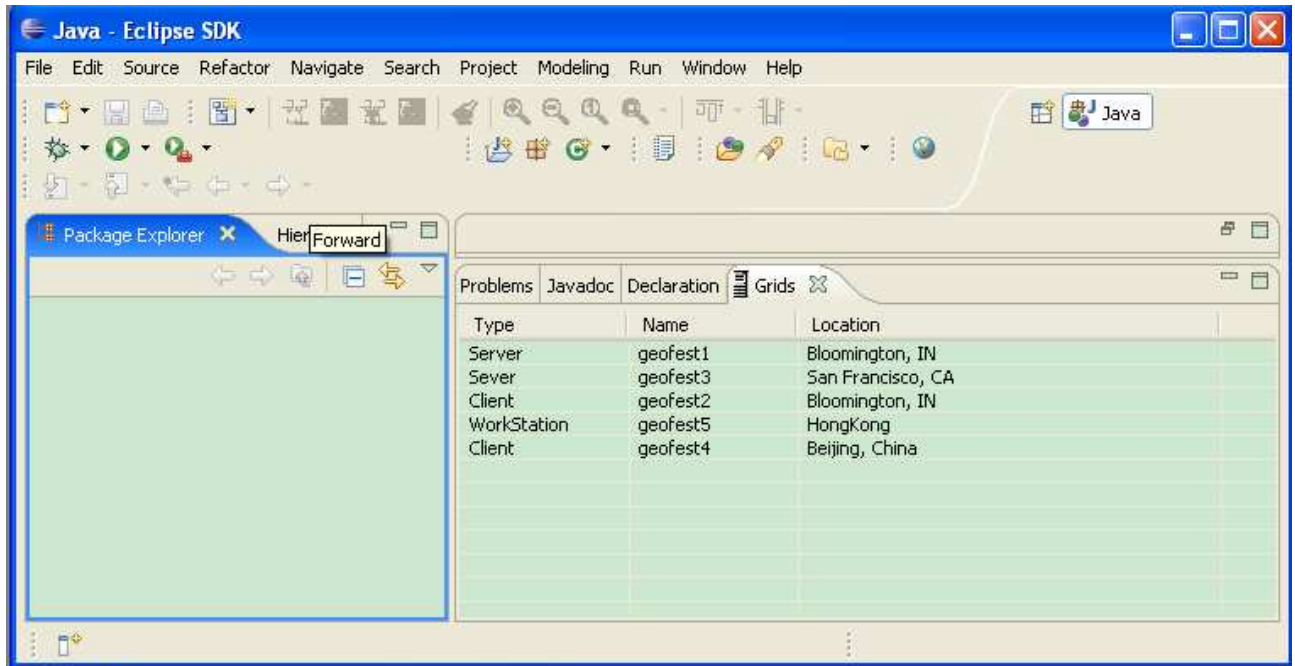


Figure 2-13 An enhanced, distributed resources viewing interface for Grid Builder

2.5 Grid Services Management Architecture and System

Characteristics of today's Grid includes but not limited to increasing complexity, components widely dispersed and disparate in nature and access, and with dynamic component failure scenarios such as nodes, network, processes. Grid services must meet general QoS and life-cycle features, and need to be managed to provide dynamic monitoring and recovery, and static configuration and composition of systems from subsystems.

- We investigated Grid Services Management architecture and system. Core features of management architecture should support remote management, firewall and NAT traversal, extensibility, scalable, and fault-tolerance.
- We installed and experimented with HPSearch 1.0.4 for Grid Services Management.
- We implemented a new Eclipse plug-in with the management system code base to keep future development in a consistency along with Grid Builder software.
- We implemented a grid service wrapper so that the resource manager can deploy and manage a resource seamlessly.

A high-level grid service management architecture for the management system is illustrated here:

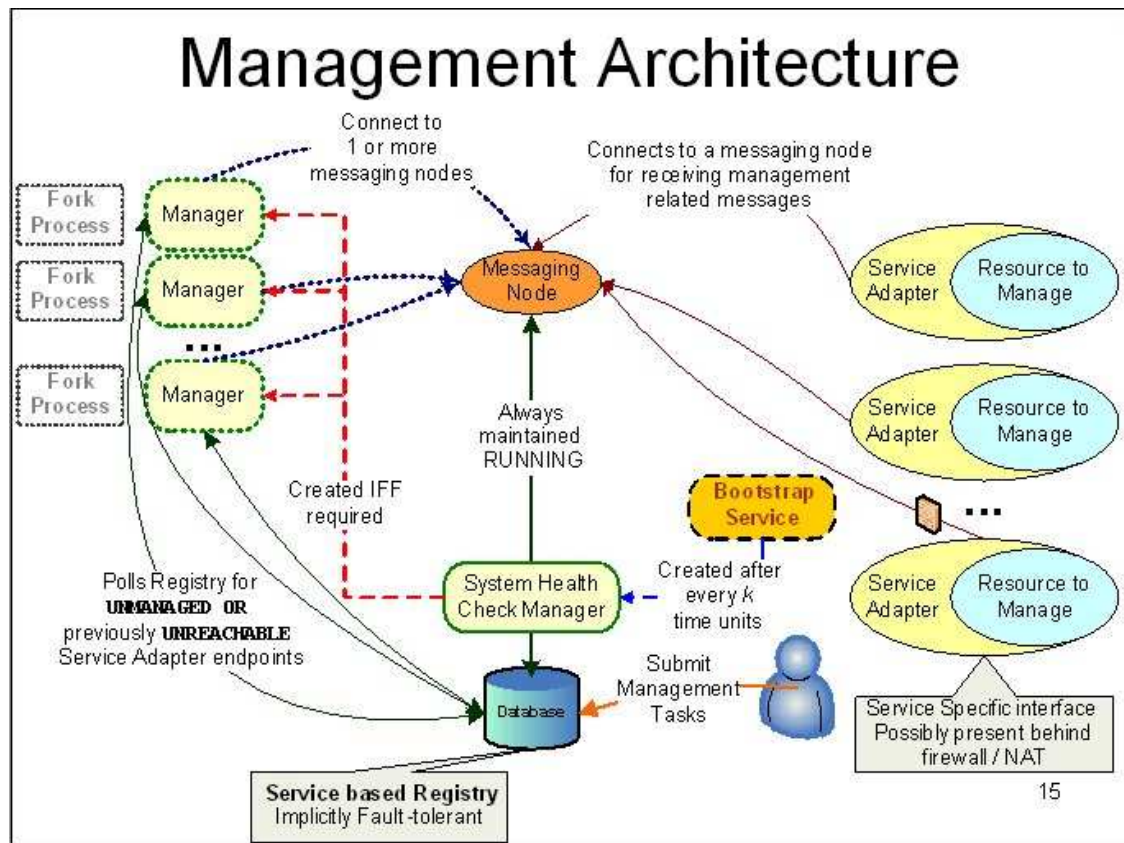


Figure 2-14 A distributed grid service management architecture

The management system was tested on both standalone and distributed nodes setting.

- For standalone (a single node) setting, all services such as fork, bootstrap, brokeradapter were running on the same machine.
- For distributed nodes, the broker nodes were running on separate machines and can be managed remotely.
- For distributed nodes setting, we tested successfully for NAT traversal capability to manage remote broker nodes.
- Currently, the management system was implemented for managing message nodes (i.e., message brokers) only.
- The service wrapper code was tested successfully with a simulated web service.

The management system was designed specifically for managing message nodes, and the interface is closely related to features of the message brokers. Application-specific functions such as *Topology Generator* may not be applied to other types of resources.

Key functions of the management system include:

- Configuration and lifecycle operations
- Global view of all accessible resources including their links
- Resource status monitor (e.g., tracking logs)
- System status maintenance (recovery, fault tolerance, etc.)
- Resource-specific features
 - Input and output interfaces
 - Unique functionalities

The integration of the management system and NCCGBT was implemented in an Eclipse-based environment by

- Extending the bootstrapping process to set up a metadata catalog for handling different types of resources
- Predictable input and output interfaces were defined by metadata
- Two schemas for each resource type
 - Essential information
 - Non-essential information (e.g., additional information)
- Metadata specifies both generic and non-generic features of a resource instance

A high-level architecture view for the management system as follows:

The Architecture at High Level

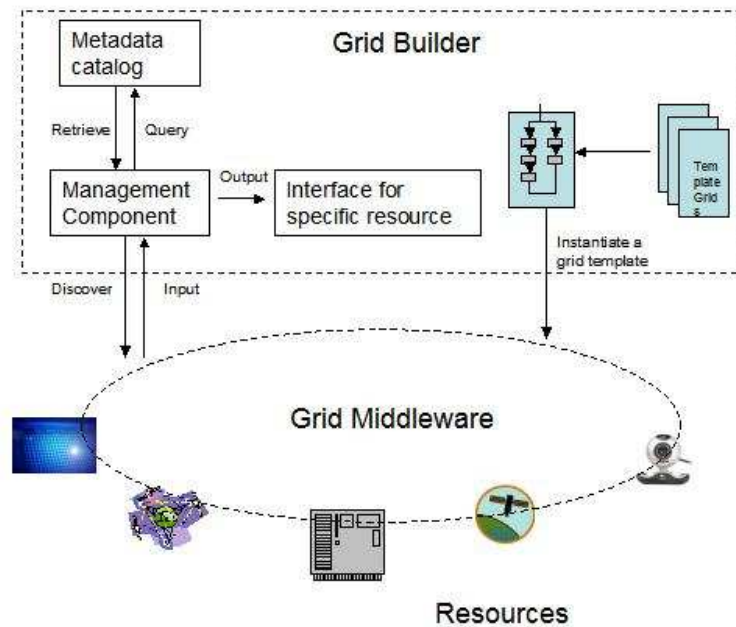


Figure 2-15 A high-level Grid Builder architecture

In the high-level design for the Grid Builder (NCCGBT), we include a metadata catalog that the management component could query and retrieve metadata from. The management component could output to interfaces for specific resources, and could be discovered by a Grid Middleware, which links to globally distributed resources and services. The management component could also receive input from the Grid Middleware.

Illustrated below is the Bootstrap Interface of the management system:

Bootstrap Interface

It is used to
discover
accessible
broker nodes

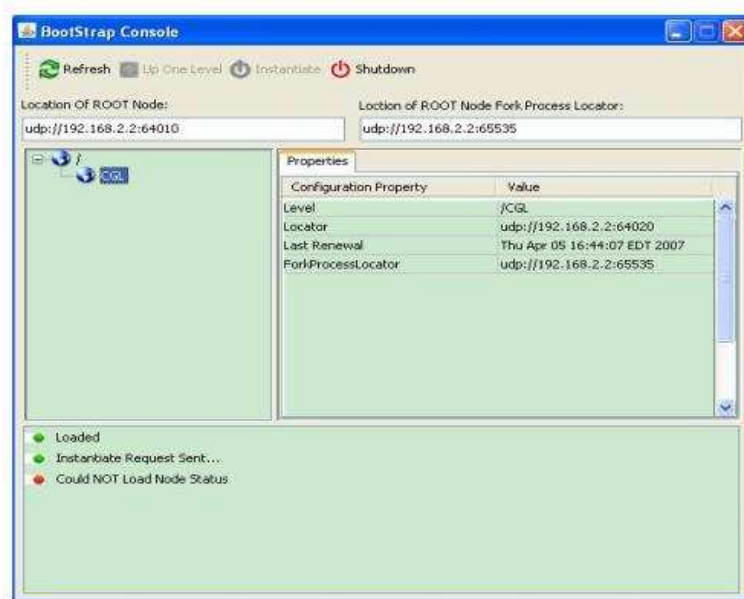


Figure 2-16 A bootstrap service interface for the Grid services management system

The main objective of the bootstrap service interface for this version of the management system is to enable easy administrative operation to discover accessible message broker nodes and view the properties of each node.

A management interface for management Narada Brokering messaging fabric is illustrated in **Figure 2-17**:

Management Interface

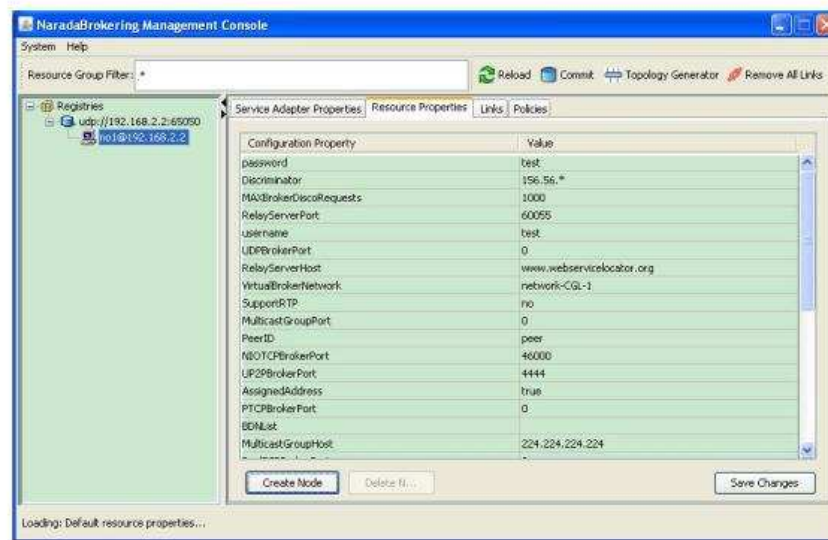


Figure 2-17 A management interface for managing a Narada Brokering fabric

Further, we completed the installation and testing of QuakeSim2 in an isolated LAN, which consisted of two nodes successfully. We

- Installed services including RDAHMM, GEOFEST, STFILTER on one node.
- Installed the offline Google Map server on the other node and configured WMSConnection servlet to download and store map images.

2.6 Summary – Generalizing and Prototyping Extended Grid of Grids Technology

2.6.1 Problem Statement

- Information and communication have played increasingly critical roles in our nation's security
- The Global Information Grid (GiG) is not one global seamless construct
 - Different pieces have different stakeholders with different missions

- Each has own name and unique vision of net-centric operations
- Many operations have been done independently
- Unable to satisfy interoperability, scalability, and security information management requirements for Net-Centric Operations without an advanced grid-based scalable service-oriented framework

2.6.2 Challenges

- Operational tasks (network management, enterprise services management, information staging, and dissemination management) need to be done across the network using common tactics, techniques and procedures
- The necessary balance of the competing demands for standardization, customization and modernization is the biggest challenge
- To integrate global grid technology with collaboration technology to provide a framework for net-centric operations to examine and derive warfighter requirements on the GiG

2.6.3 General Goals

- Build Net-Centric Core Enterprise Services in fashion compatible with GGF/OGF and industry
- Add key additional services including those for sensors and GIS
- Support System of Systems by federating Grid of Grids supporting a heterogeneous software production model allowing DoD greater sustainability and choice of vendors
- Build tools to allow easy construction of Grid of Grids

2.6.4 Research Objectives

- Develop Net-centric Collaboration Grid Middleware (NCGGCM)
- Develop components for Grid of Grids capability
- Develop a Net-centric Collaboration Grid Builder Tool (NCCGBT)
- Prototype commercialization potential for DoD
- Demonstrate non-DoD related commercialization potential

2.6.5 Research Methodology

- Our solution builds upon existing technology and infrastructure currently being developed across the grid and web services communities
- The major innovation is a systematic mapping between NCOW Core Enterprise Services and Grid and Web Services Architectures

2.6.6 Research Approach

- Analyze Net-Centric Operations and Warfare (NCOW) service specifications and relate core enterprise services in Net-Centric Enterprise Services (NCES) to core OGF and Web Services (WS-*) standards
- Develop the Grid of Grids architecture and information management middleware to address federation of legacy and new DoD enterprise systems with service-oriented mediation between component collaboration, sensor, information and computing grids
- Develop prototype for NCES capabilities (Collaboration, Messaging, Management, Security/Information Assurance, Discovery, Mediation, User Assistance, Storage, Applications) with advanced Grid and Web Service standards support
- Develop static and dynamic Net-Centric Collaboration Grid Builder Tool compatible with Web service workflow standards
- Demonstrate for Earthquake science and DoD applications

2.6.7 Research Tasks

The R&D effort is divided into five major tasks:

1. Implementation of Collaboration Grid Middleware
2. Enhanced NCOW Core Enterprise Services (NCES) with Enterprise Control services and Metadata services
3. Design and implement of Grid of Grids mediation algorithms and NCOW services
4. Design and implement of Net-Centric Collaboration Grid Builder Tool
5. Technology Demo

2.6.8 Part 1 Implementation Status

- Grid Builder tool, which is compatible with Web service workflow standards
 - Grid template
 - BPEL workflow designer
 - Resource Viewer
- Management System
 - Discover messaging nodes
 - Status monitor
 - System status maintenance (recovery, fault-tolerance, etc.)
- Template Grids designed for ServoGrid/QuakeSim modeling and simulations

- Integration of the Management System and Grid Builder

2.6.9 CTS 2007 Demonstration

We have developed an extensible framework for managing resources (services included). We used BPEL to represent workflows and demonstrate a BPEL workflow designer. A workflow engine could be integrated for executing workflows. The management system can provide useful information such as load balancing during deployment.

We demonstrated using ServoGrid/QuakeSim. The Earthquake Grid is an example of a “Grid of Grids”.

It is a representative Web Service Grid application which includes

- Web services: provide access to data and codes
- Portlet: acts as an aggregation of client interfaces
- We can build web services from sketch or use those built by others (e.g. legacy services)
- Data can be retrieved from archives or from real-time filters

QuakeSim2 provides services such as:

- AnalyzeTseri Service
- AntVisco Service
- Gnuplot Service
- RDAHMM Service
- STFilter Service
-

The prototype system gives an overview of the earthquake grid and allows the user to select services/grids based on situation assessment

A view of the QuakeSim2 portal with multiple portlets and services is show below:

QuakeSim2 Demo

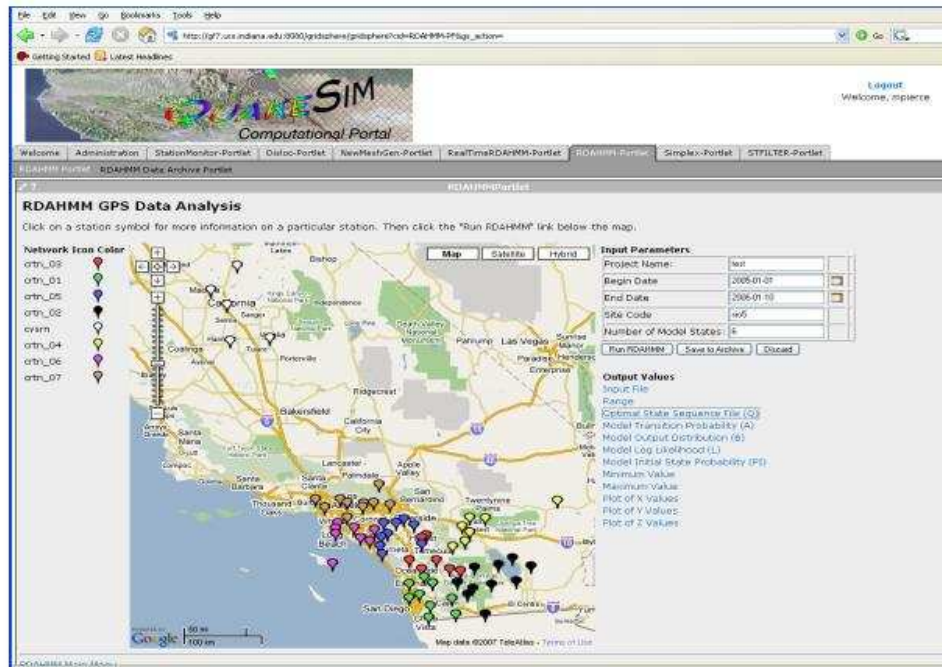


Figure 2-18 An illustration of the QuakeSim2 portal interface

2.6.10 The Implication of The Demonstration

- The end user, who serves as the administrator, can select a grid template based on the current situation assessed
- The extended workflow designer enables the user to edit the template and resource requirement
- The selected grid will be deployed on available resources
- The management system keeps monitoring resource status
- The user can access services through portals which are customizable
- Distributed, different services/grids are federated and interoperable in a seamless way

The R&D results of Part 1 – Generalizing and prototyping extended Grid of Grid technology offers a solid foundation for the follow-on work to design and develop a Sensor-Centric Grid of Grids Middleware Management System and SCGMMS Grid Builder.

3 Sensor-Centric Grid of Grids

Increased use of sensors in commercial and military environments is being driven by the need for better intelligence data and by advancement in technology, which provides smaller, less costly and more capable sensors. It is, however, not sufficient and in many situations not productive to just provide lots of sensor data to decision-makers at all levels for their missions on hand. It is valuable to have a framework that supports seamless integration of loosely-coupled COTS and custom-developed sensor data analytic, management, visualization and presentation tools, and real-time collaboration capability for sharing situational awareness.

3.1 Project Goal

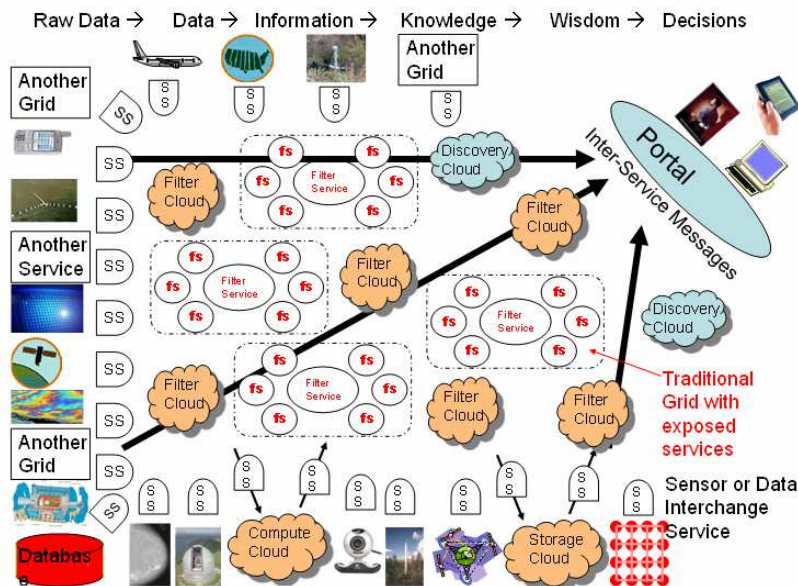


Figure 3-1 A conceptual any time, anywhere, anything Grid of Grids system

Sensor Grid Architecture

Above in Figure 3-1 we picture the Grid system that we aim at. The system consists of a Grid of Grids of Services. The Grids are either opaque such as the storage, compute and filter clouds marked or a collection of explicitly advertised services as in traditional grids. Services and Grids send and receive messages while sensors respond to control messages and send raw data in messages.

The main objective of the Sensor-Centric Grid Middleware project is to design and develop an enabling framework to support easy development, deployment, management and real-time visualization and presentation of collaborative, geo-coded sensor-centric grid applications with flexibility, extensibility and scalability for situational awareness.

The framework, called Sensor-Centric Grid Middleware Management System (SCGMMS) is based on an event-driven model that utilizes a publish and subscribe communication paradigm over a distributed message-based transport network. A key capability component in SCGMMS is a Grid Builder (GB) module which supports the assemblage of grids and resources - a compositional model of assembling a multitude of subgrids and relevant resources into a mission-specific grid application.

For the current prototype, one illustrative application – called Impromptu collaborative sensor sharedlet - based on the assemblage of two important subgrids, namely a real-time multimedia collaboration grid and hierarchical, executable sensor grid was developed.

A specific design objective for the Impromptu collaborative sensor sharedlet is to provide an intuitive user interface to facilitate client-side UDOP (User Defined Operation Picture) and COP (Common Operation Picture) features, which are essential for agile formulation and sharing of visual, situational awareness and effective decision-support.

Much of the system requirement for SCGMMS is driven by the needs of sensor-centric, UDOP-capable client applications for situational awareness.

The Sensor-Centric Grid Middleware Management System (SCGMMS) is discussed in the remaining of Section 3. The system and darchitecture to support User-defined Operational Pictures (UDOP) in SCGMMS is discussed in Section 4. A live, advanced technology demonstration of the SCGMMS prototype for a sample sensor-centric situational awareness application, which utilized a variety of globally deployed physical and computational sensors, including some carried by remotely operated robots will be discussed in Section 5.

The latest SCGMMS prototype was deployed globally during the International Symposium on Collaborative Technologies and Systems 2008 (CTS 2008) in Irvine, California for a real-world, advanced and live technology demonstration. The live demonstration comprised of 3 locations – Irvine (California), Bloomington (Indiana) and Hong Kong. Details of the advanced technology demonstration is discussed in Section 5.

The system has been packaged and documented in Appendix A - User Guide for Sensor Developers, Appendix B - User Guide for Sensor-Centric Application Developers, Appendix C - User Guide for System Administrator, Appendix D - User Guide for Sensor Administrator, and Appendix E - User Guide for SCGMMS Application User.

During the course of the Phase 2 projecct, we used some inexpensive commercially available sensors including RFID for our R&D activities. We invented a new, algorithm for RFID positioning with initial result than some popularly used algorithms in our test cases. This new RFID positioning algorithm, documented in Appendix F - could be further explored and developed into a computational sensor service for a sensor grid in some follow-on projects.

3.2 Sensor-Centric Grid Middleware Management System Architecture

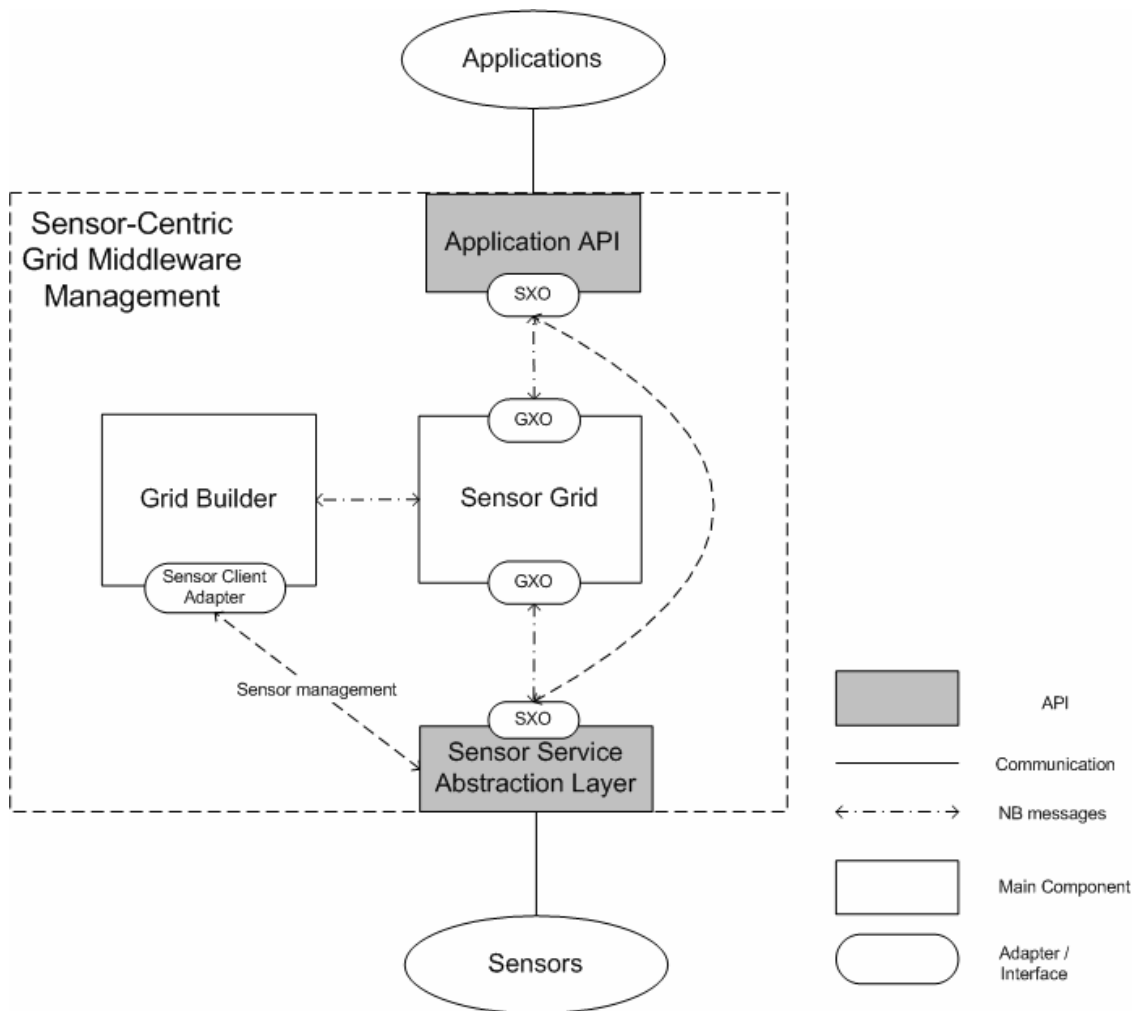


Figure 3-2 SCGMMS overall architecture

Sensor-Centric Grid Middleware Management System (SCGMMS) 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 **SCGMMS API** (see section 3.5 for details), 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 SCGMMS's **Sensor Service Abstraction Layer (SSAL)** (see section 0 for details).

NaradaBrokering (NB) is the transport-level messaging layer for SCGMMS. It is a distributed message-based transport network with a publish-subscribe messaging model.

By using NB as the transport different components of SCGMMS can be deployed and works collaboratively in a distributed manner.

The overall architecture of SCGMMS is shown in Figure 3-2. Internally SCGMMS is composed of 2 main modules – **Sensor Grid (SG)** and **Grid Builder (GB)** which serves different functions.

3.2.1 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 scale as number of managed sensors increases
3. Fault tolerant - failure of transports OR management components should not cause management architecture to fail

Details of GB is discussed in Section 3.3.

3.2.2 Sensor Grid (SG)

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

3.2.2.1 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 for changes. Sensor data normally does not pass through SG except that it has to be recoded intentionally. In this case data of that particular sensor is subscribed by SG.

3.2.2.2 Application/Sensor Grid flow

Applications communicate with SCGMMS 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 forwards messages among the relevant sensors and that application. SG must always check which sensors meet the selected filter criteria and update the list of relevant sensors accordingly. It then sends an update message to application if there are any changes of the relevant sensors.

3.2.2.3 Grid Builder/Sensor Grid flow

Sensors' properties are defined in GB. Applications have to 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 Grid Builder.

3.2.2.4 Application/Sensor flow

SG provides each application with information of sensors they need according to the filtering criteria. The application then communicates with sensors through the Application API for receiving data and sending control messages.

Details of SG is discussed in Section 3.4.

3.2.3 Sensor-Centric Grid Middleware Management System (SCGMMS) API

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

Details of SCGMMS API is discussed in Section 3.5.

3.2.4 Sensor

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

3.2.4.1 Sensor Client Program

A sensor needs a **Sensor Client Program (SCP)** to connect to SCGMMS. The SCP is the bridge for communication between actual sensors and SCGMMS. On the sensor side SCP communicates with the sensor through device-specific components such as device drivers. On the SCGMMS side SCP communicates with SCGMMS through Sensor Service Abstraction Layer (refer to section 0 for details).

Figure 3-3 shows a physical sensor and the corresponding Sensor Client Program.

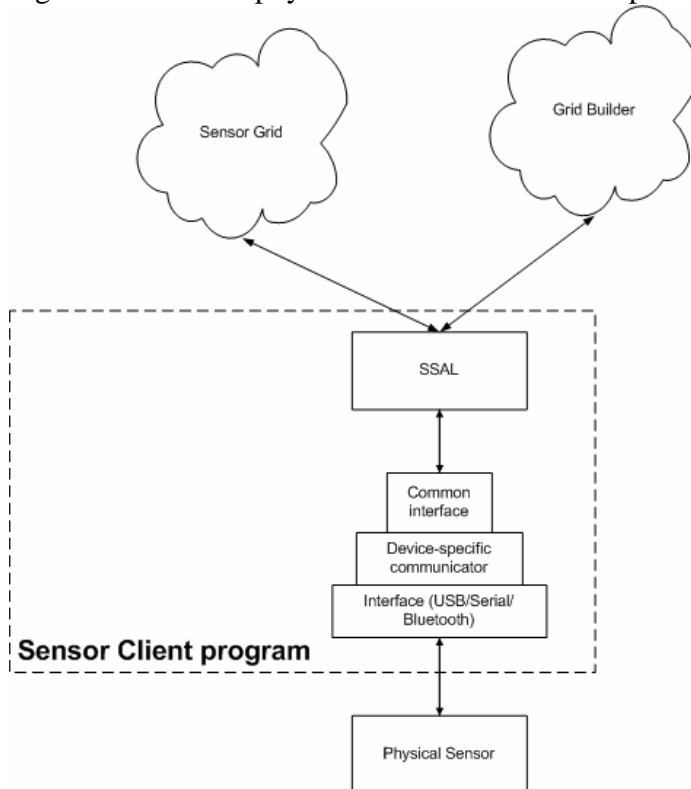


Figure 3-3 Structure of A Sensor Client Program

3.2.4.2 Computational Service

Computational Service is a special kind of sensor which does not take input from the environment. Instead, they take output of other sensors as their input, perform various computations on the data, and output the processed data finally. It is called a sensor because it totally matches with our definition of sensor.

Figure 3-4 shows the data flow of how environmental data is transformed by processing data through a sensor and a Computational Service. The architecture of SCGMMS allows the data source to be assigned and reassigned dynamically.

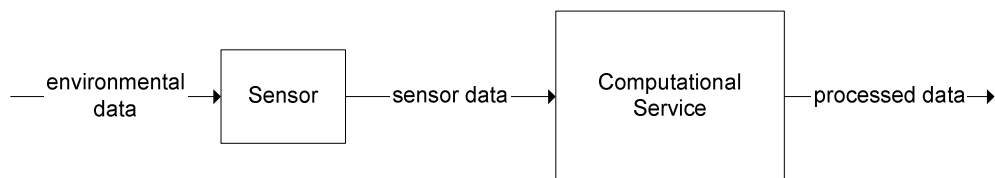


Figure 3-4 Computational Service

3.2.4.3 Supported sensors

To illustrate the usage of SCGMMS, several predefined sensors are supported in our initial implementation.

Hardware sensors

1. GPS device
2. RFID reader and tags' signal strength
3. NXT Robots with ports attached to 4 of the following sensors:
 - a. Light
 - b. Sound
 - c. Touch
 - d. Ultrasonic
 - e. Compass
 - f. Gyro
 - g. Accelerometer
 - h. Temperature
4. Wii Remote Controller
5. Nokia N800 Internet Tablet PC video camera
6. PC Webcam

Computational services

1. Video Edge Detection Sensor (software sensor)

3.2.5 Sensor Service Abstraction Layer (SSAL)

SCGMMS can potentially support large amount of sensors of different kind. Ease of adding new sensors by different sensor developers without internal knowledge of SCGMMS is one of the most important requirements. SSAL provides a common interface for adding new sensors to the system easily. Sensor developers have to write simple programs utilizing SSAL libraries for connecting sensors to SCGMMS. Afterwards the sensor will be available for all applications right away.

Details of SSAL is discussed in Section 3.5.

3.3 Grid Builder

3.3.1 An Overview of the Grid Builder Architecture

Architecture Overview

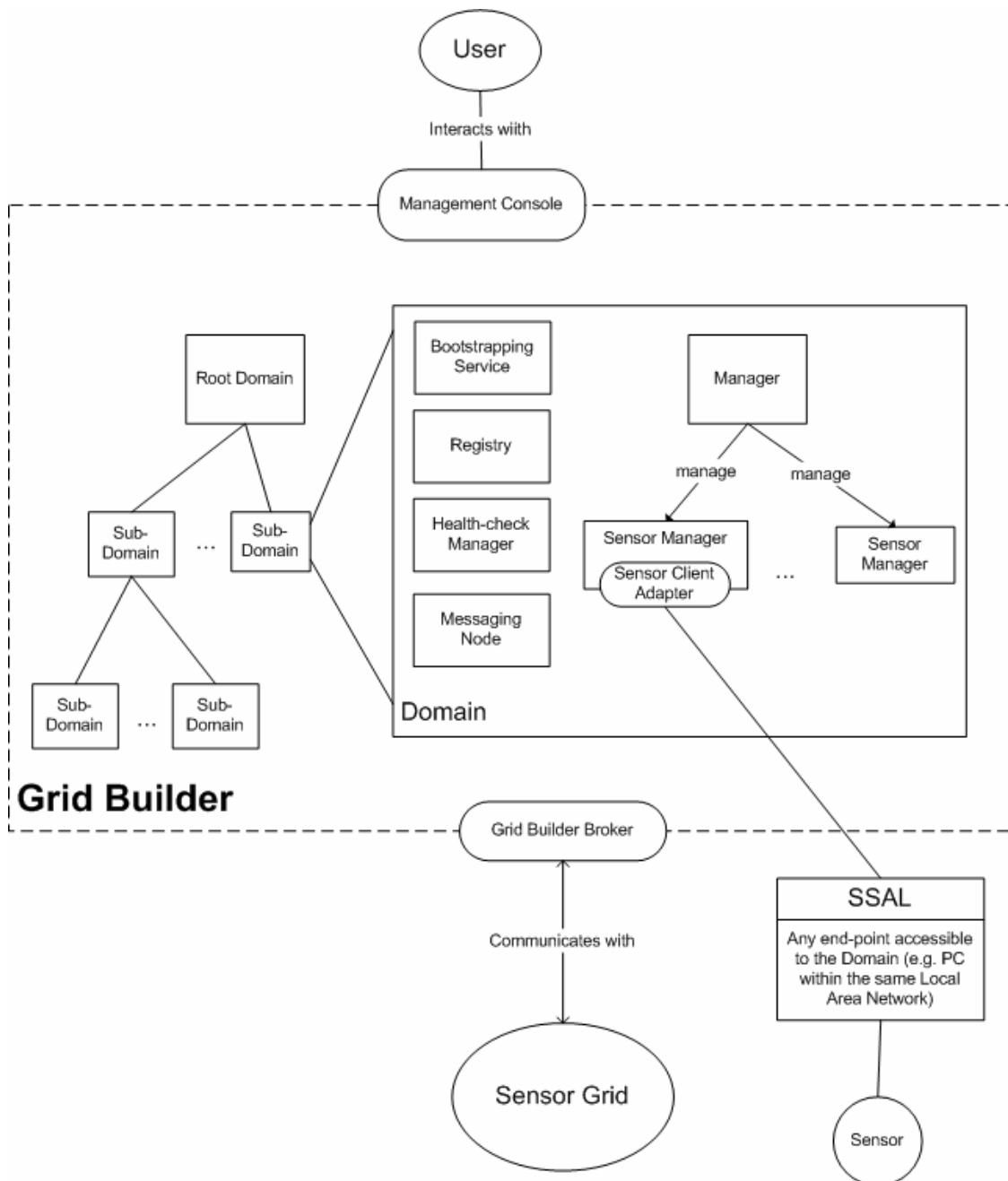


Figure 3-5 An overview of the Grid Builder architecture

Figure 3-5 depicts the overall Grid Builder (GB) architecture. GB is originally designed for managing Grid-of-Grids. For this project, GB is extended to include the management of a generalized sensor-centric grid of grids. Description of GB will focus on this specialized version. CGL-developed hpsearch is adopted and extended for this work [2].

The Grid which GB manages is arranged hierarchically into **Domains**. Each domain 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**. Each domain is started by its **Bootstrapping**.

Within each domain, there exist some basic components:

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 our 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 Sensor Manager how the SSA is to be managed, and defines the **properties** of the sensor bound to the SSA.

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).

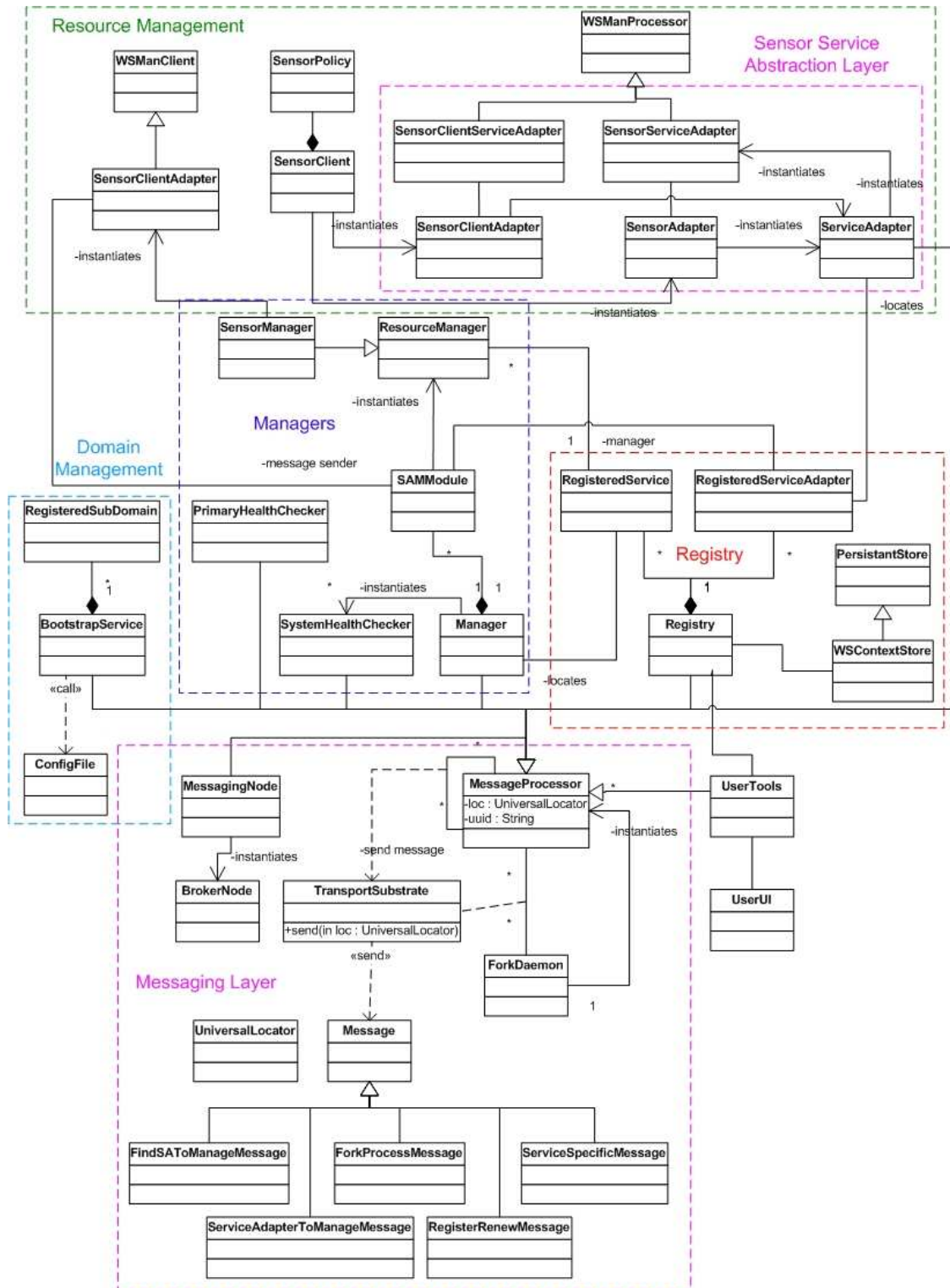
Bootstrapping Service

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.

3. 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.

3.3.2.1 Class Diagram



The diagram shows the class diagram of significant classes in GB. They are categorized into 5 main categories:

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.

GB has a set of classes dedicated for message passing. Each module has a unique UUID and one or more UniversalLocator(s) (UL). UL provides all the information necessary to identify a module in the network, including transport type, host address, port and path. 4 transport types are supported: UDP, TCP, 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.

Modules which want to receive message should implement the MessageProcessor interface and associates 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 (mgmySystem.conf).

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 heart beat for some time.

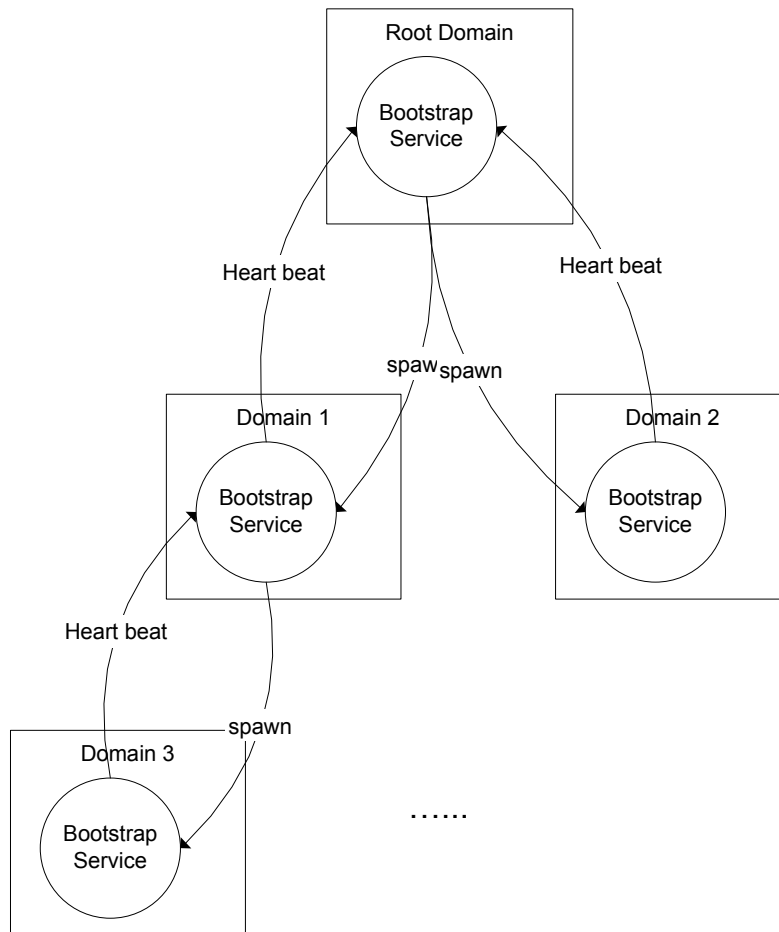


Figure 3-7 Domain Management

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 create a SAMModule 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.

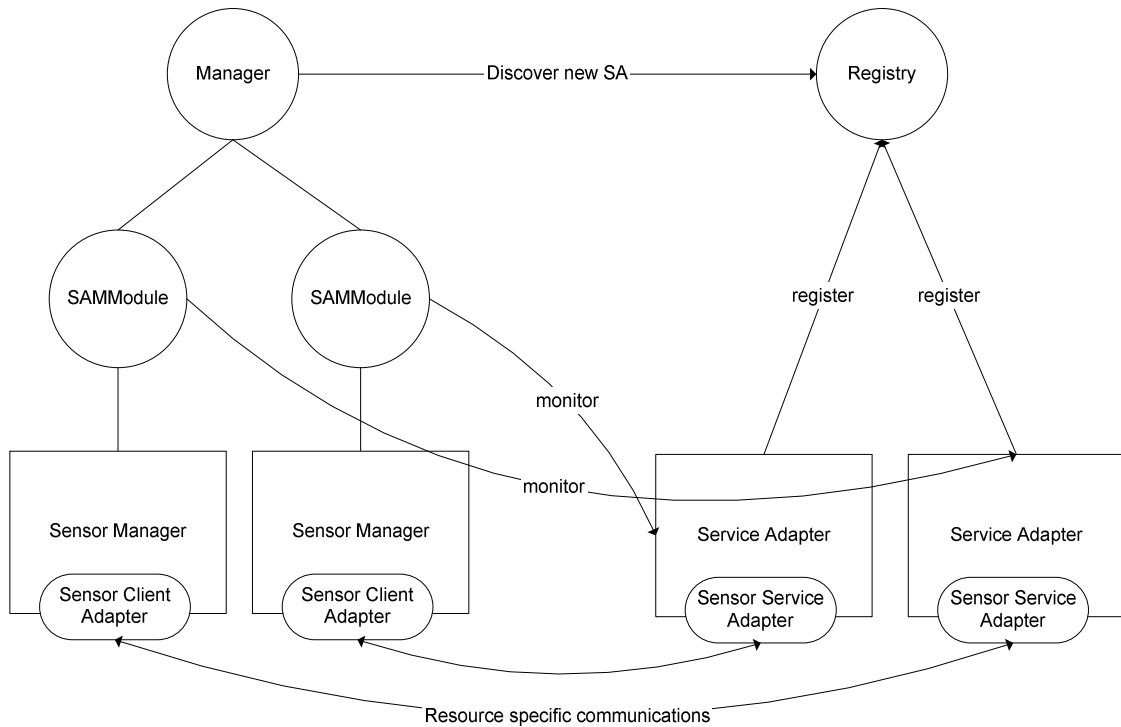


Figure 3-8 Manager and Service Adapters

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 SCGMMS.

Sensor Service Abstraction Layer (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. The whole SSAL involves classes of SXO as well.

Communication between resource managers (i.e. SensorManager) and Resources (i.e. SensorServiceAdapter (SSA)) uses the Web Service (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 [3].

Figure 3-9 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.

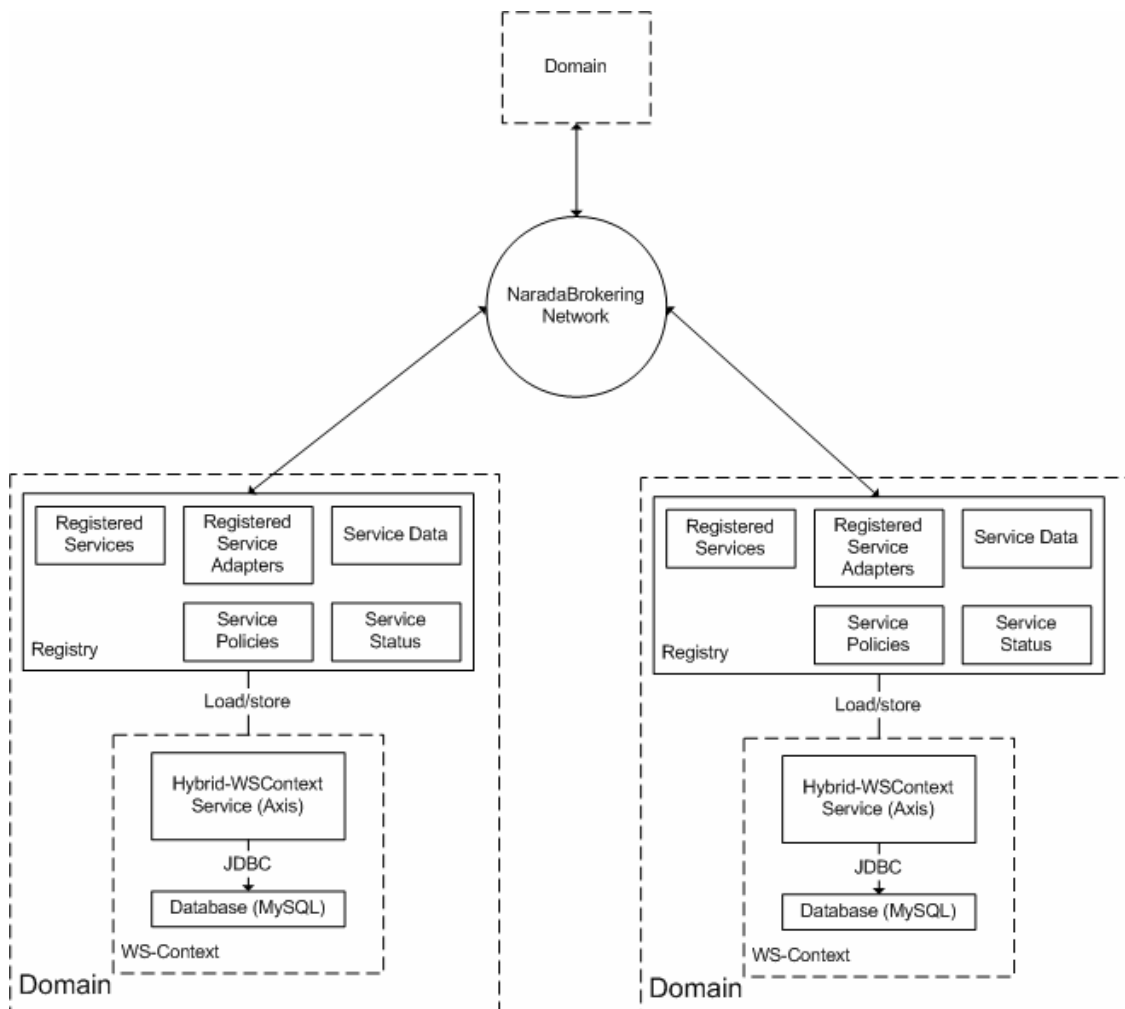


Figure 3-9 Registry and WS-Context

3.3.2.2 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:	UserUI
Package name:	cgl.hpsearch.NaradaBrokering.usergui
Description:	Graphical user interface of GB's management console

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:	Responsible 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 resource 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()

3.3.3 Important Features

3.3.3.1 System Health Check

Every module in GB are deployed in a distributed manager and linked together by different network protocols. A health check system is therefore fundamental to ensure every modules are indeed deployed and working properly. GB performs periodic **System Health Check (SHC)** to ensure that every thing is up and running.

SHC can be divided into three stages:

Initialization

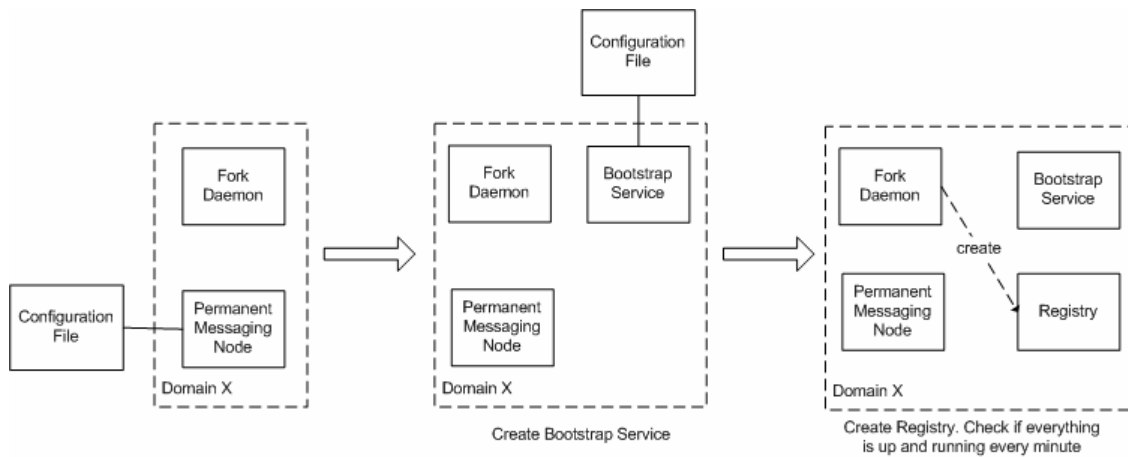


Figure 3-10 System Health Check (SHC) Initialization

To start a new Domain X, a user has to execute 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.

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 3.3.4.3).

Detect Changes

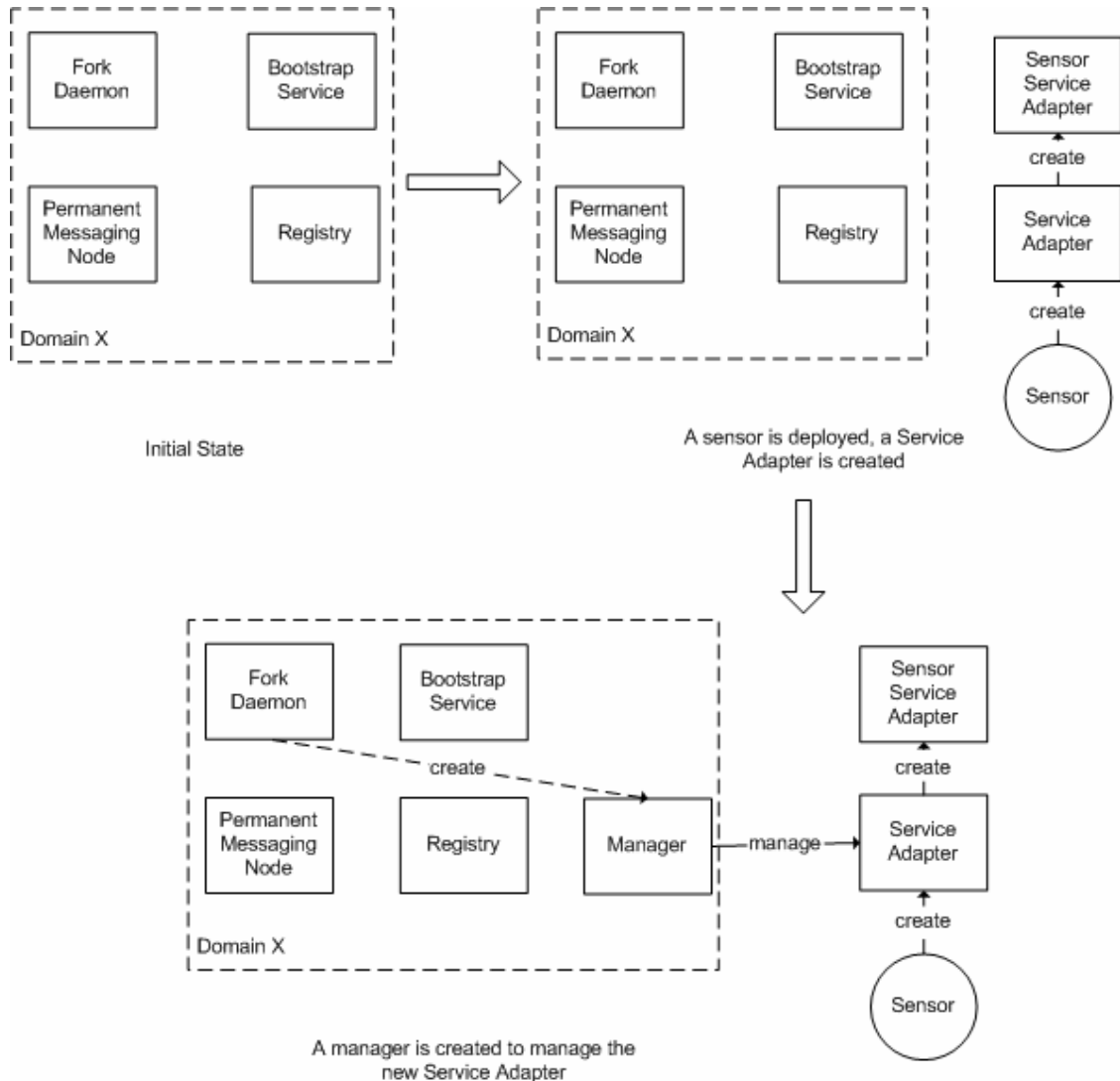


Figure 3-11 Adding Service Adapter

When we introduce changes 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 Service Adapter for the sensor which in turn creates a Sensor Service Adapter. If no Manager is present in the domain, a Manager process is created by ForkDaemon to manage the sensor through Service Adapter.

Maintain System State

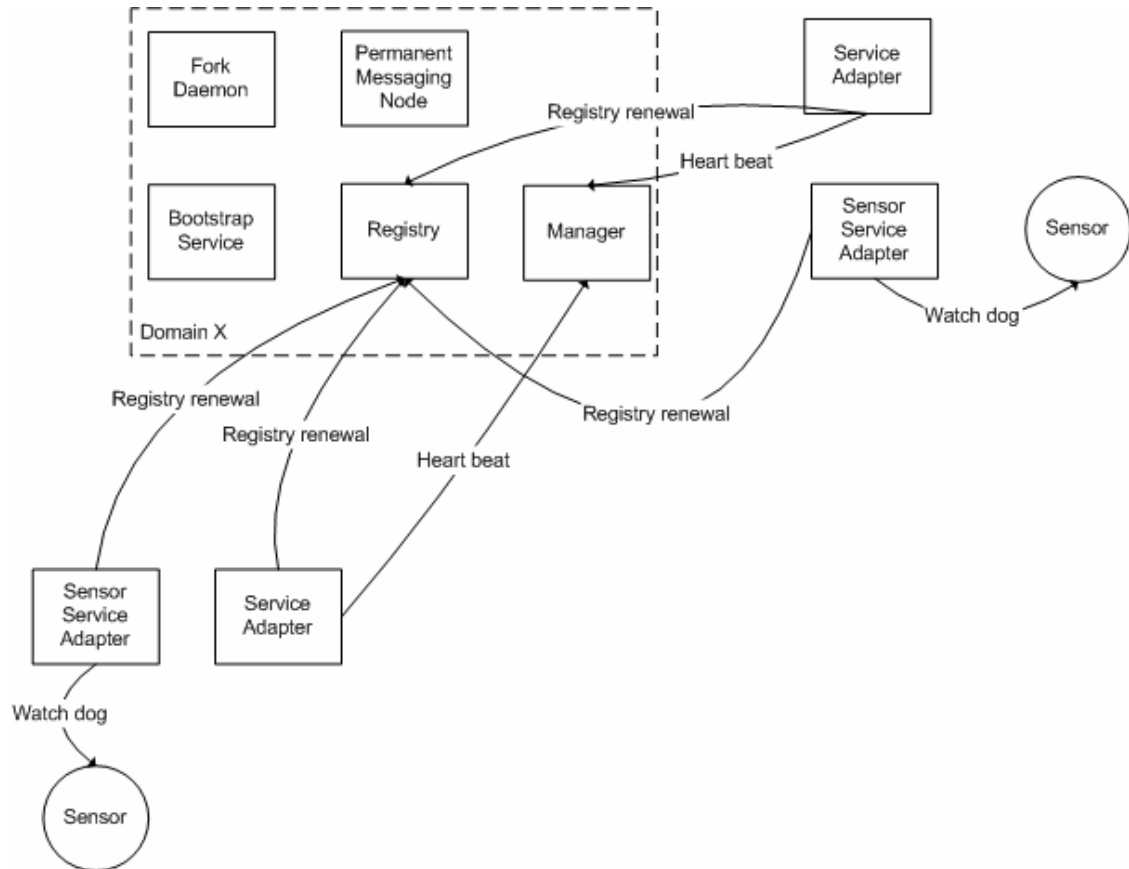


Figure 3-12 System Health Check (SHC) Maintaining System State

To make sure that every resource is up and running, each module periodically notifies its manager and the registry of its presence.

3.3.3.2 Classification Scheme

Classification defines all properties which are shared by all sensors supported by SCGMMS. 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 3-6 shows the class diagram of classification. It can be divided into 3 categories:

Sensor Property

In order to introduce a new sensor to SCGMMS, the following properties have to be defined in class SensorProperty:

Table 3-1 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

SCGMMS 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 SCGMMS 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 xml format. Class SensorClassificationUtil provides operation for message serialization and deserialization.

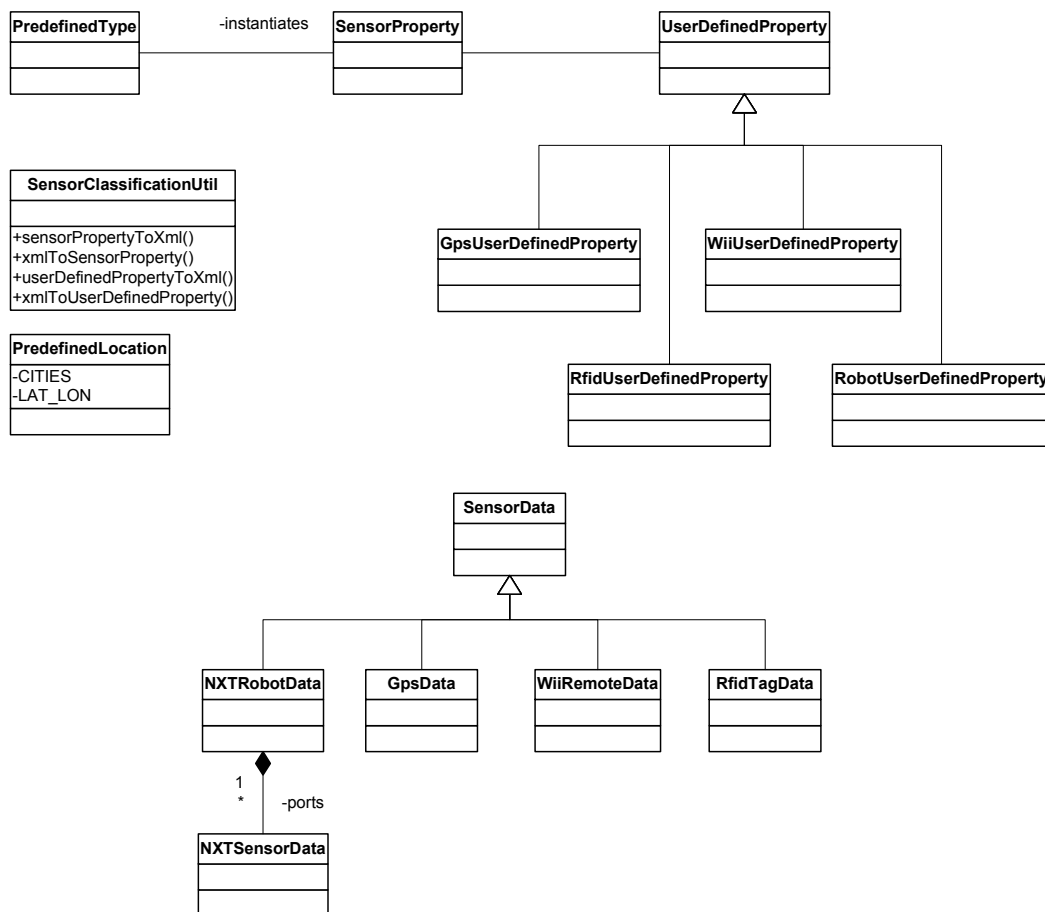


Figure 3-13 Class diagram of classification scheme in SCGMMS

3.3.3.3 Filtering Mechanism

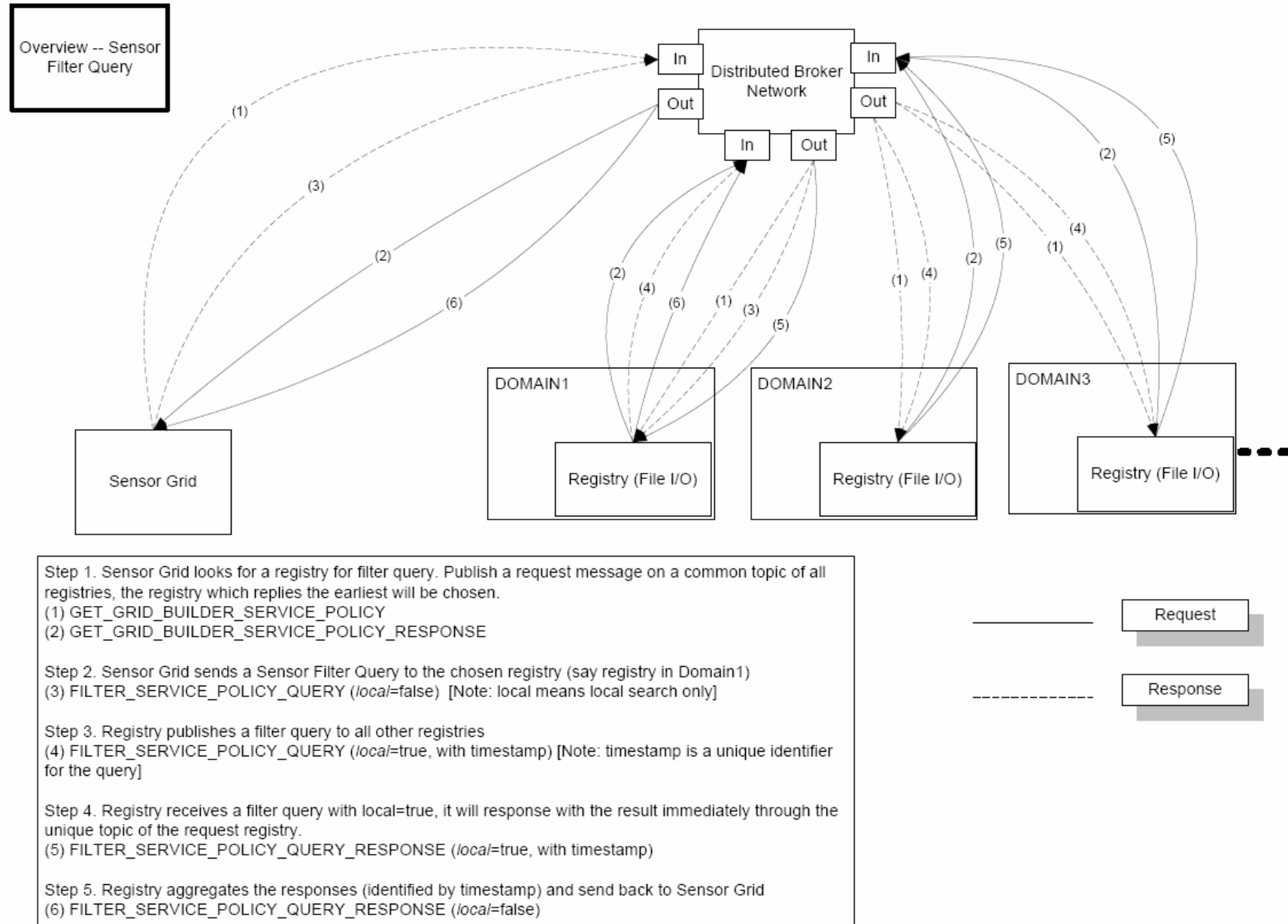


Figure 3-14 SCGMMS sensor filtering mechanism in a distributed architecture

At the application standpoint 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.

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 3.3.3.2 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 with exact match in string comparison with ALL the properties defined in the filter should get through. Similarly sensors which have properties with 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 SAID 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

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.

3.3.4 Detailed Description

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

3.3.4.1 Starting a Domain

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

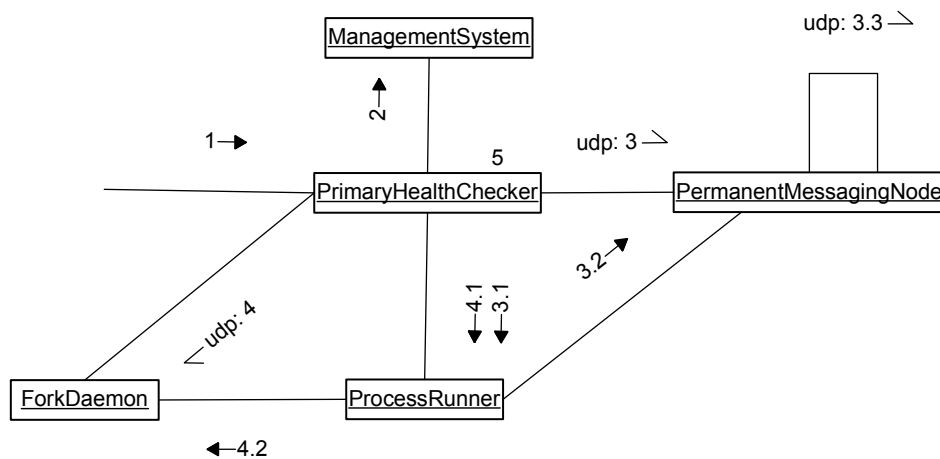


Figure 3-15 Event flow when starting a sensor grid domain

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

3.3.4.2 Starting BootstrapService of a Domain

When a domain is start, it undergoes the following Bootstrap sequence.

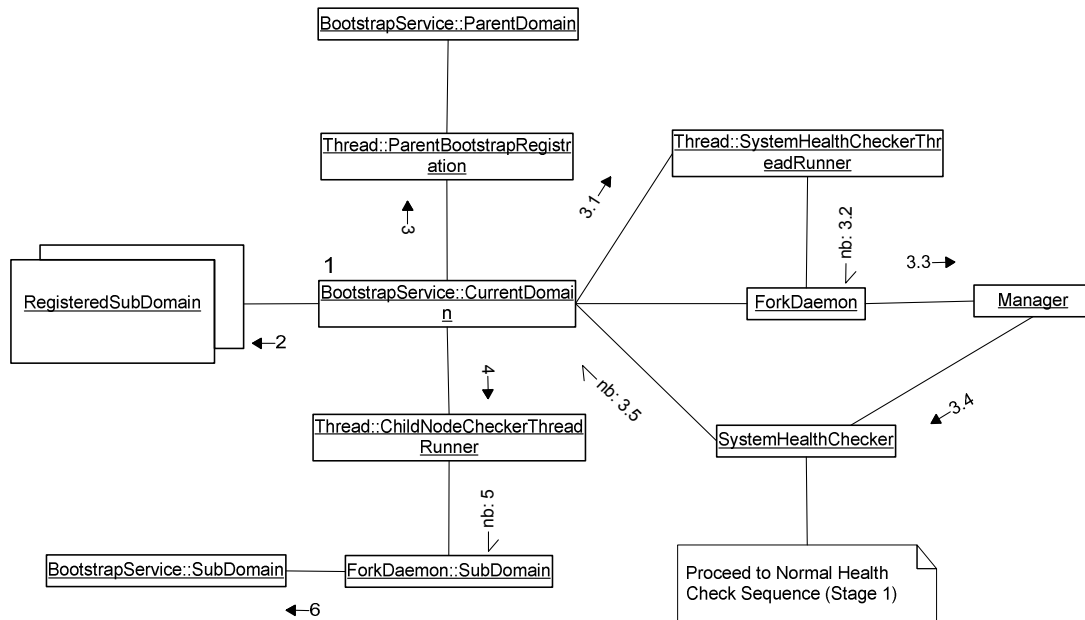


Figure 3-16 Starting BootstrapService of a Domain

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 3.3.4.3 for 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, spawns 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

3.3.4.3 Normal Health Check Sequence (Stage 1)

System Health Check has a number of stages. During the first state, Bootstrap Service checks if the Registry is present. If not, creates a Registry process using the Fork Daemon.

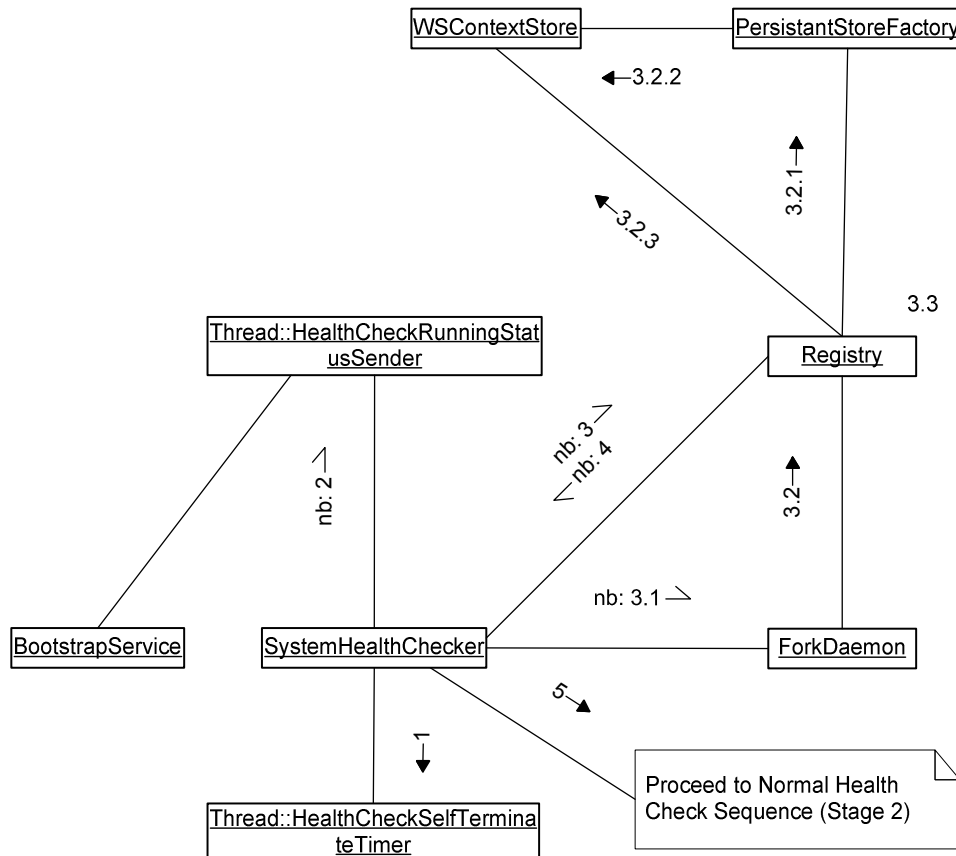


Figure 3-17 Normal Health Check Sequence (Stage 1)

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. Please proceed to 3.3

- 3.2.1. Registry asks PersistentStoreFactory 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
- 3.3. Registry initializes NB transport by subscribing to two topic – 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. Proceed to section 3.3.4.4 .

3.3.4.4 Normal Health Check Sequence (Stage 2)

System Health Check has a number of stages. During the second stage, Bootstrap Service checks if enough Managers are spawned as defined in the configuration file.

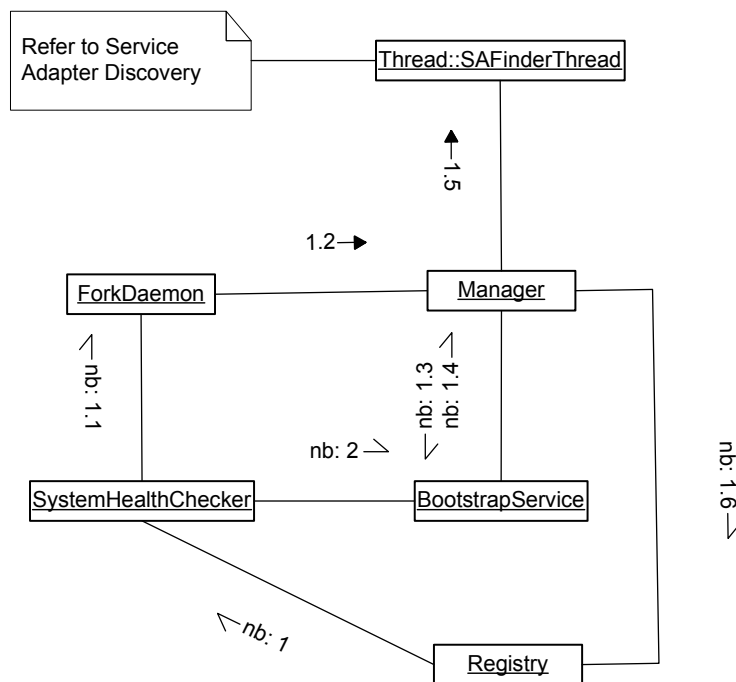


Figure 3-18 Normal Health Check Sequence (Stage 2)

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 interval. For details of this part, please refer to section 3.3.4.6 .
- 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 3.3.4.3) and tries spawning the process(s) missing

3.3.4.5 Registered Service Adapter Health Check Sequence

SAMModule notifies the Service Adapter which Manager it should send heart beat messages to

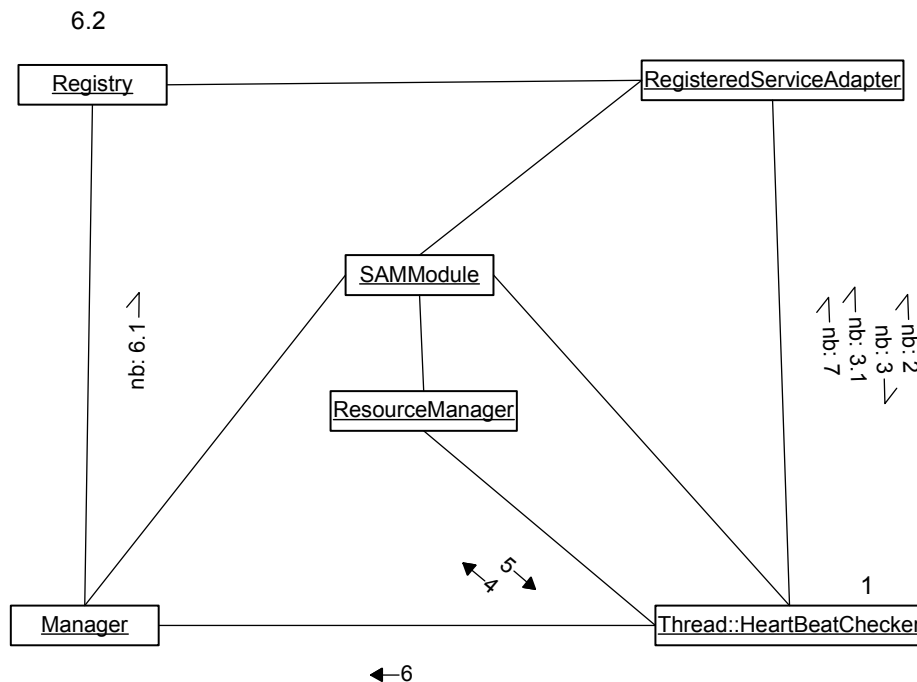


Figure 3-19 Registered Service Adapter (RSA) Health Check Sequence

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

3.3.4.6 Service Adapter Discovery

System Health Check checks if every Service Adapter is associated with its Manager.

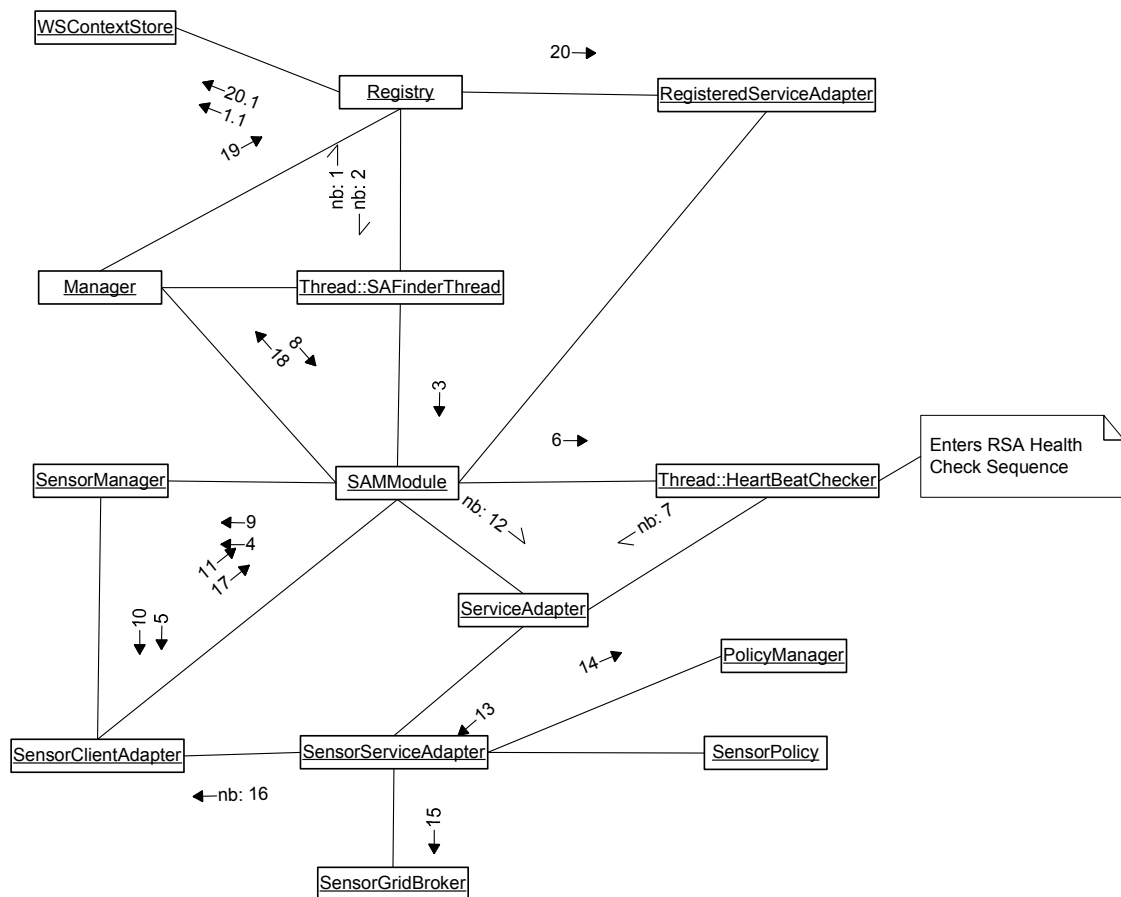


Figure 3-20 Message flow of service adapter discovery in a sensor grid

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 WSCContextStore
- 1.2. Registry replies with ServiceAdapterToManageMessage to the Manager if there is at least one ServiceAdapter (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
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 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 SensorServiceAdapter (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 WSCContextStore
 - 19.2. The RSA is stored in memory

13. Subscribes to a new NB topic according to the returned instancelid. 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 session 3.3.4.4 .

3.3.5.2 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 below shows the message flow of disconnecting a sensor through GB's management console.

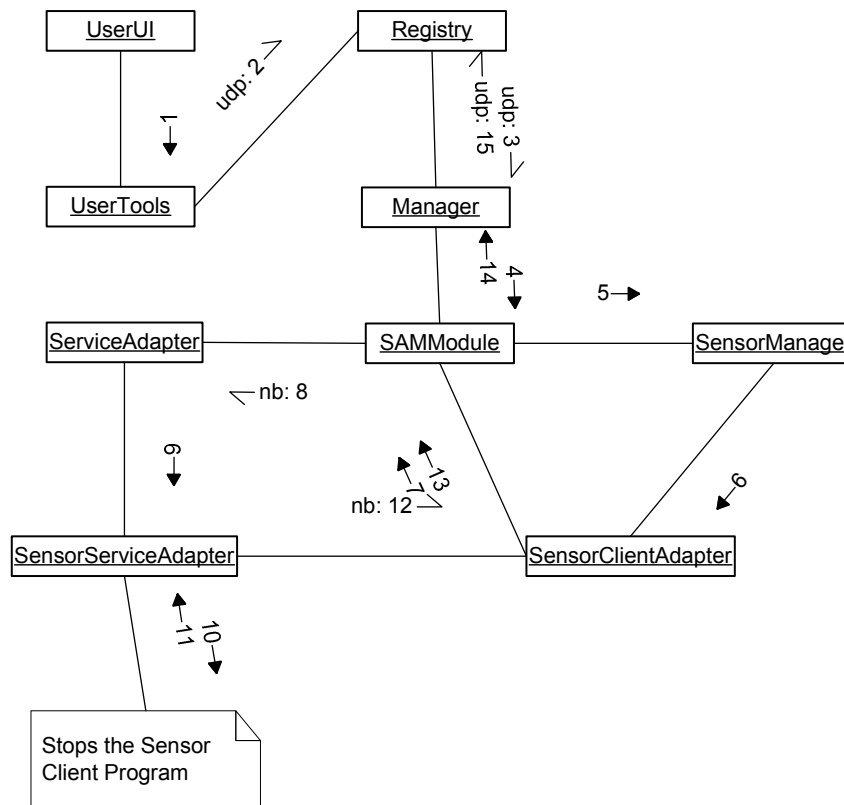


Figure 3-22 Disconnecting a sensor by using the Grid Builder management console

- 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 SensorServiceAdapter (SSA)
- 10 SensorServiceAdapter stops the sensor through SSAL. For details please refer to section 3.4.4.9
- 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

3.4 Sensor Grid

3.4.1 Overall Architecture of Sensor Grid and Related Modules

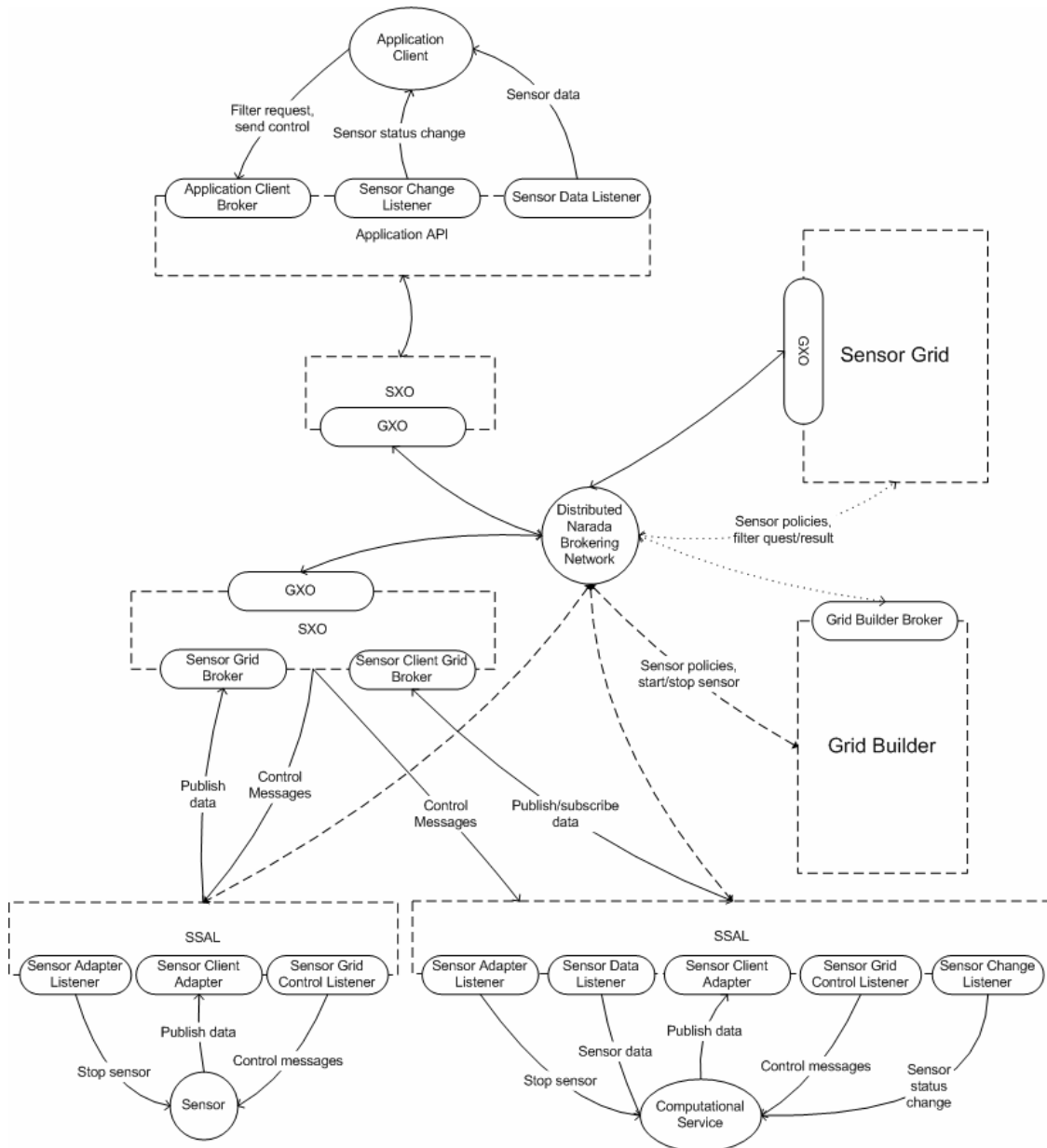


Figure 3-23 Overall Architecture of Sensor Grid and related Modules

Sensor Grid (SG) is the brokering module of SCGMMS connecting the sensors, application clients and Grid Builder. It serves two functions:

3.4.1.1 Message Brokering

It 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.

GXO

GXO is a messaging layer which uses NaradaBrokering (NB) for message passing. It has the following characteristics:

1. supports a lot of transport layer protocols, including tcp, niotcp, udp, http, https and so on
2. abstracts messages into byte, text and object messages which performs automatic message serialization and de-serialization
3. uses a topic-based, publish and subscribe model which eliminates the need for identifying end points explicitly
4. allows flexible construction of brokering network

With the use of GXO, messages can propagate to the destination with minimum programming effort.

SXO

SXO is a layer built on top of GXO. It is the internal API which facilitates communications between sensors, application clients and SG. It handles the connection and disconnection of both sensors and application in a seamless and fault-tolerant manner. It contains logic and libraries for both Application API and SSAL to communicate with applications and sensors respectively.

Application API

All kinds of applications communicate with SCGMMS through the same API. The Application API provides libraries for applications to:

1. access data and metadata of sensors
2. send control messages to sensors
3. notified for change of sensor status
4. send filter requests to SCGMMS

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

Application Client Broker

Interface 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 used by application clients to receive messages from SG such as sensor status change.

Sensor Data Listener

Interface used by application clients to receive data from sensors.

To support different applications, Application API in turn communicates with SCGMMS through SXO. For more detailed description of Application API, please refer to section 3.5.

SSAL

All sensors communicate with SCGMMS through SSAL. Remember each sensor has a corresponding Sensor Client Program (SCP) to communicate with SCGMMS. SSAL provides libraries for sensors to do the following through SCP:

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

Not all kind of sensors have to use all functionalities 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 below:

Sensor Client Adapter

An interface for publishing data

Sensor Data Listener

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

Sensor Adapter Listener

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

Sensor Change Listener

An interface for being notified when the subscribed sensor has any status change. Used by Computational Service

Sensor Grid Control Listener

An interface which sensors listen to control messages

For more detailed description of SSAL please refer to section 3.5.2 .

3.4.1.2 Application Management

In SCGMMS, 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. The figure below shows a scenario which 2 applications and 5 sensors are connected to SG. The four tables shows 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 concerned by the application do not hold unnecessary resources. A table is maintained to remember this mapping

Application to filter mapping (Table S_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 Grid Builder before deployment. The sensor policy is obtained from GB and cached whenever a sensor is being deployed.

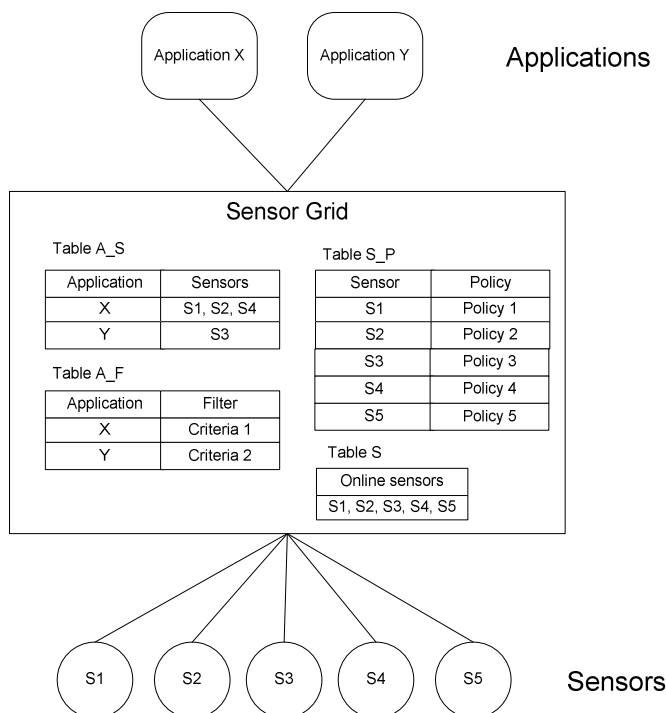


Figure 3-24 SG System Management

3.4.2 Significant Classes

3.4.2.1 Class Diagram

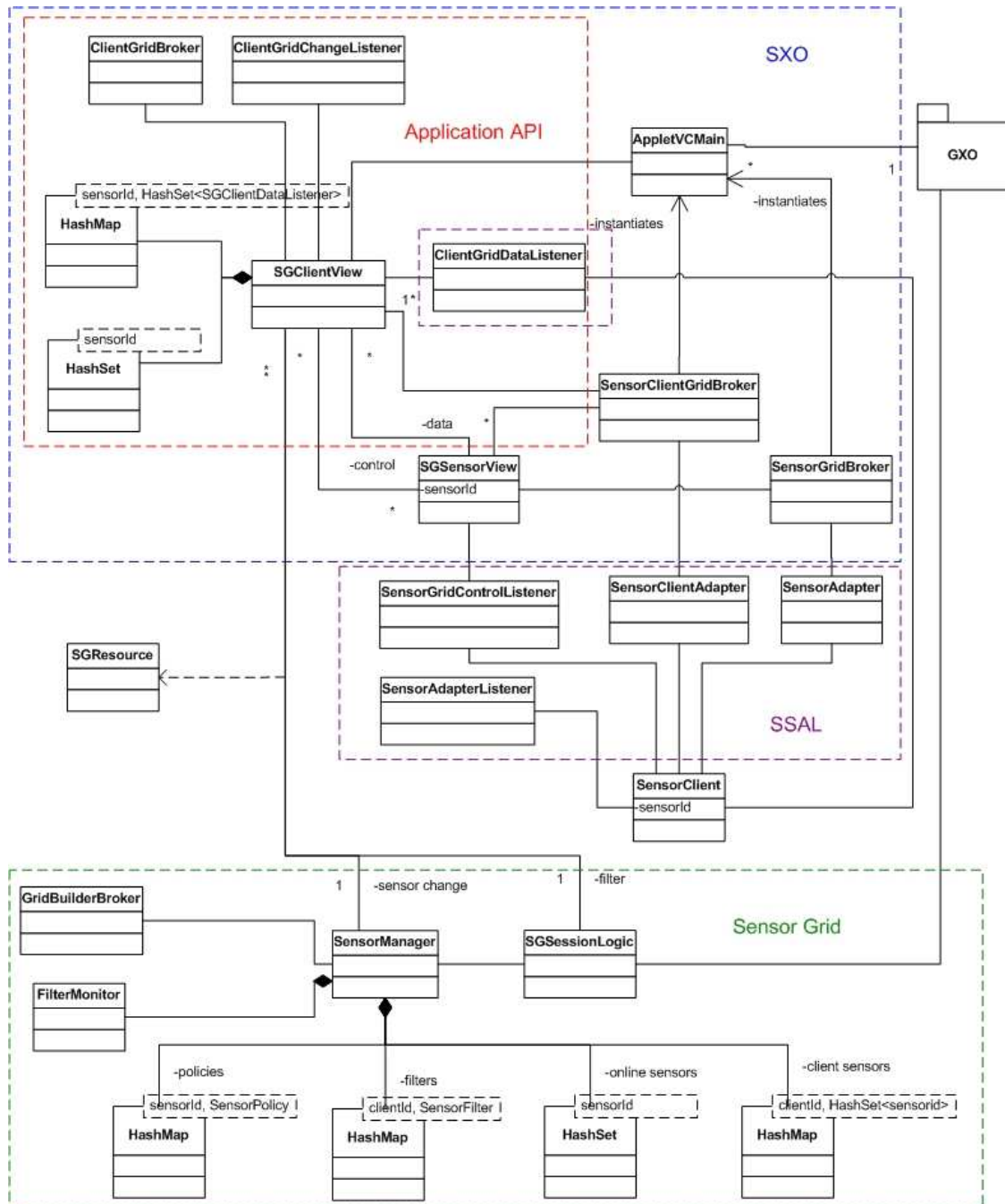


Figure 3-25 Class Diagram of SG, Sensor and Application Client

The figure above shows the class diagram of significant classes in S XO and S Sensor Grid. Within S XO, classes used by application clients and classes for sensors are also indicated respectively.

3.4.2.2 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. Notifies GXO for application joining
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:	Part of SXO. 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:	Part of SXO. 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
Important interface:	setControlListener(), publishData()
Class name:	SensorGridBroker
Package name:	com.anabas.sensorgrid.sensor
Description:	Part SXO. Brokers communication between SSAL, SG and sensors. Notifies GXO for sensor deployment and disconnection
Important	publishData(), close()

interface:

Class name: SensorClientGridBroker
Package name: com.anabas.sensorgrid.sensorclient
Description: Part of SXO. Brokers communication between SSAL, SG and service sensors. Notifies GXO for sensor deployment and disconnection

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: Actually this is an inner class of SensorManager responsible for periodic checkup to update the set of sensors for each application according to their corresponding filter

Important interface: 0

Class name: SensorManager
Package name: com.anabas.sensorgrid.session.sharedlet
Description: Part of SG. Contains the logic for managing all connected applications and sensors. Maintained 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()

Class name: SGSessionLogic
Package name: com.anabas.sensorgrid.session.sharedlet
Description: Part of SG. Responsible for handling communications with all applications and sensors through GXO. Performs state update

	through SensorManager for every connections and disconnections of sensors and applications (notified by GXO)
Important interface:	userJoined(), userLeft()
Class name:	AppletVCMain
Package name:	com.anabas.sharedlet.appletframework
Description:	Part of GXO. Resides at client side (applications and sensors) for allocating and releasing resources
Important interface:	allWindowsClosed()

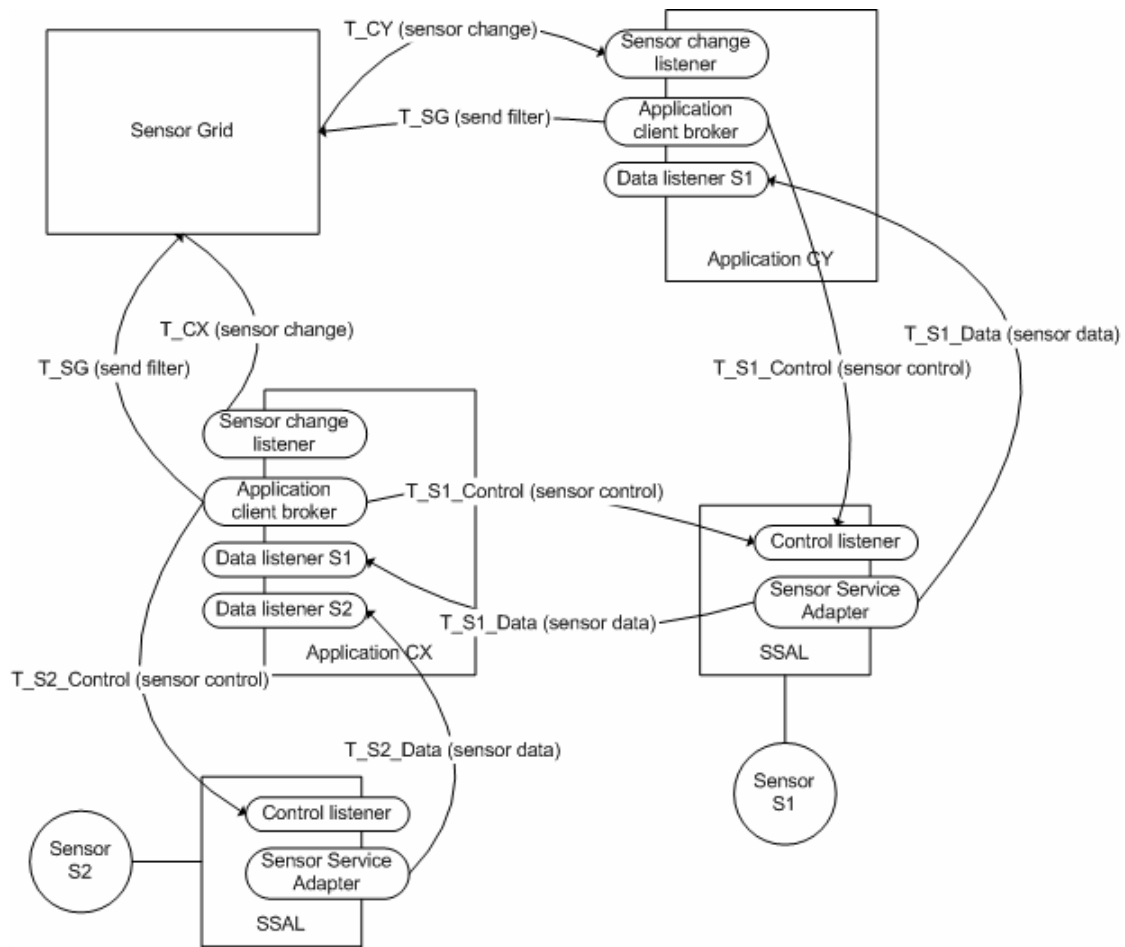
3.4.3 Important Features

3.4.3.1 NB Data Flow and Topic Management

Communication between applications, sensors and SG relies on NaradaBrokering (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.



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

Figure 3-26 Message flow between a Sensor Grid (SG), applications and sensors

3.4.4 Detailed Description

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

3.4.4.1 Sensor Grid Startup

Sensor Grid starts a perpetual session.

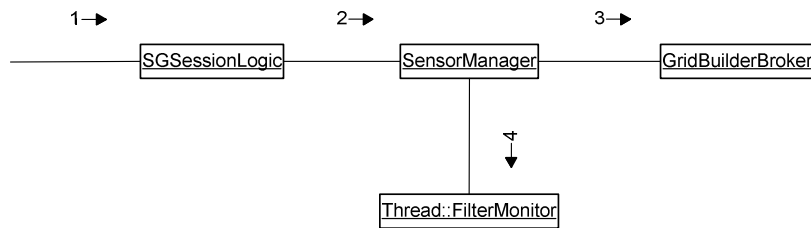


Figure 3-27 A Sensor Grid startup sequence

- 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 Grid Builder
- 4 A thread is created which do filtering for different application-clients for every 5 seconds

3.4.4.2 Deploying a Sensor

When deploying a sensor through the Grid Builder, sequences of messages are invoked to enable the management of deployed sensors as well as mechanisms to filter sensors based on sensor policies. Message flow when a sensor is deployed through Grid Builder is illustrated in Figure 3-28

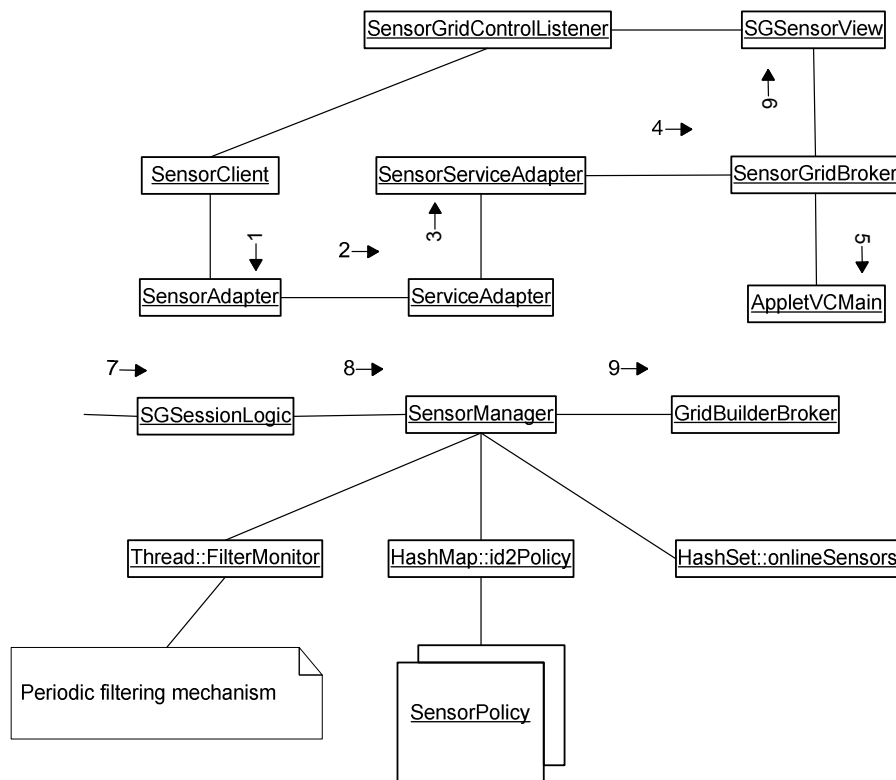


Figure 3-28 Message flow when deploying a sensor through the Grid Builder

- 1 The sensor client program instantiates SensorAdapter when it is started by Grid Builder
- 2 SensorAdapter instantiates ServiceAdapter, which is later on managed by Grid Builder
- 3 Service Adapter instantiates SensorServiceAdapter, which resides in SSAL for communication with SensorManager of Grid Builder
- 4 SensorServiceAdapter instantiates SensorGridBroker, which communicates with Sensor Grid
- 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. Sleep for 5 seconds.
- 6 A SGSensorView 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 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 down the sensor in HashSet and its Policy in HashMap
- 9 Asks Grid Builder 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. Please refer to section 3.4.4.3 for details

3.4.4.3 Periodic Filtering

SG periodically checks the status of sensors and whether there are changes for each filter defined by applications. Below shows the message flow.

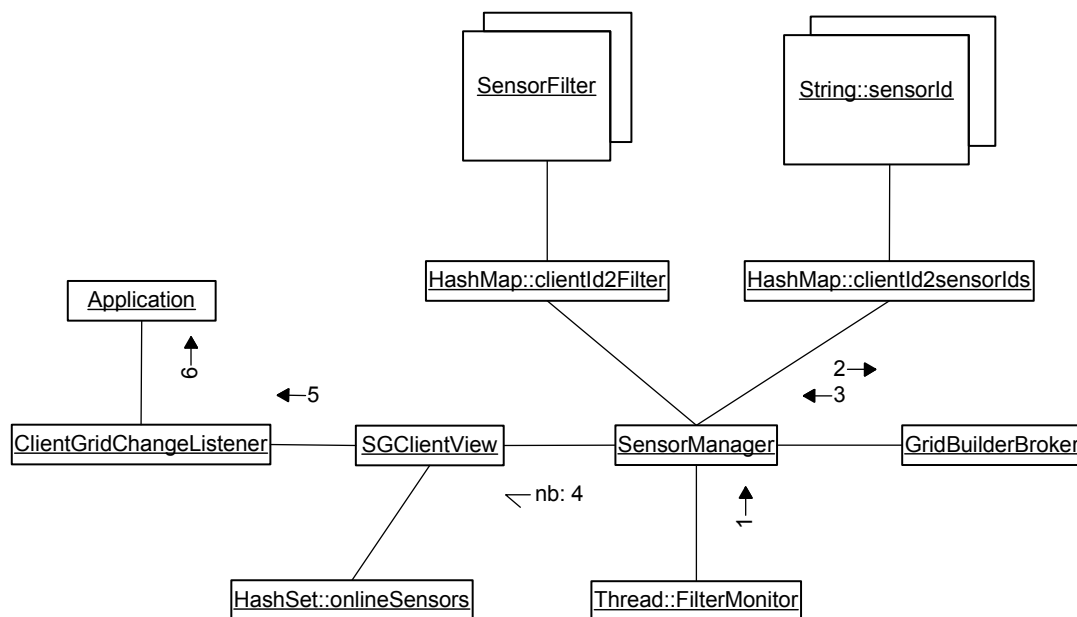


Figure 3-29 Sensor Grid message flow during periodic sensor filtering

- 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 Grid Builder 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

3.4.4.4 Application Client Joining A Sensor Grid (SG)

When a sensor grid application client joins a sensor grid (SG), the message flow is illustrated as follows:

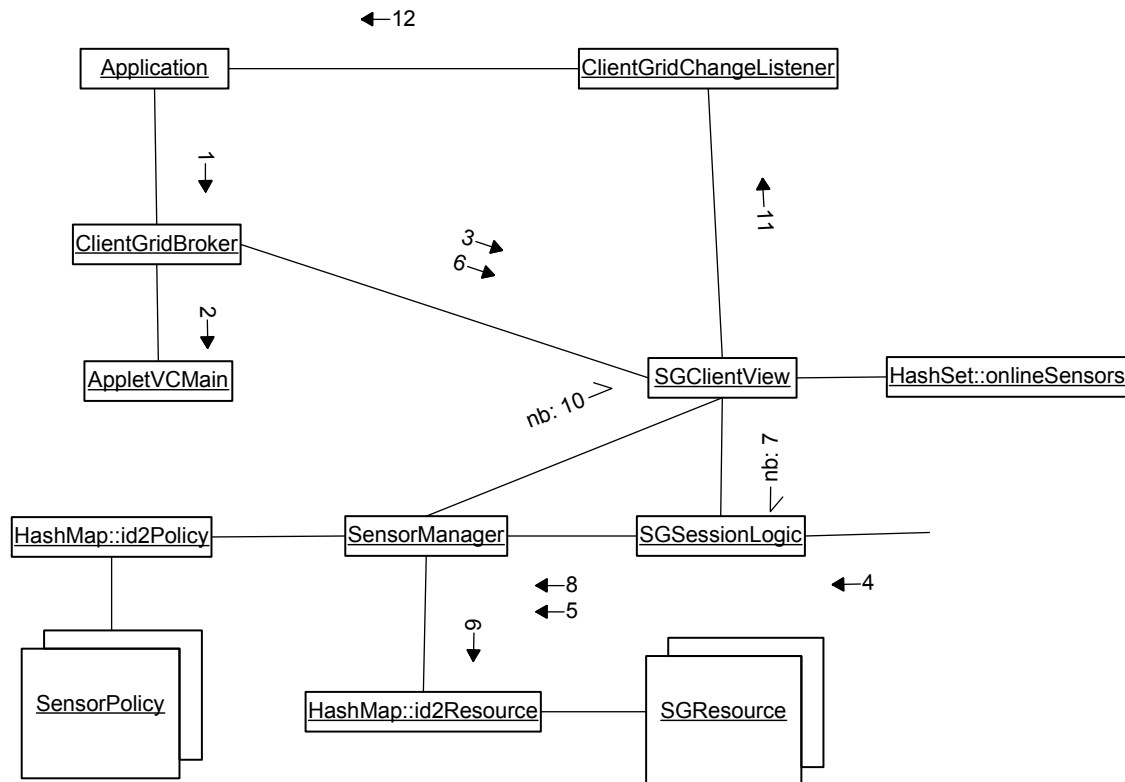


Figure 3-30 Message flow when an application joins a sensor grid

- 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. Sleep for 5 seconds.
- 3 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
- 4 The framework notifies that a new application client has joined through the SessionListener interface of SGSessionLogic (userJoined()).
- 5 invokes addClient() of SensorManager. SensorManager initializes NB streams for communication with application client
- 6 Registers application client's ClientGridChangeListener. Invokes SGClientView's startConnection()
- 7 Sends a START_CLIENT message with its client id
- 8 Forwards the request to SensorManager

- 9 Creates a HashMap which maps the id of all online sensors to SGResource instances wrapping the policy and status of the sensors
- 10 Sends a INIT_SENSOR message to the client, containing the created HashMap
- 11 Updates the cached list of online sensors in HashSet. Invokes handleSensorInit() of the registered ClientGridChangeListener
- 12 ClientGridChangeListener notifies application client of sensor change. Application client performs corresponding actions

3.4.4.5 Sensor Publishing Data

After a sensor is deployed in a sensor grid, real-time stream of sensor data and metadata will be published to the sensor grid. The message flow of a sensor publishing data to the sensor grid in which application clients could subscribe to such live streams is illustrated in Figure 3-31.

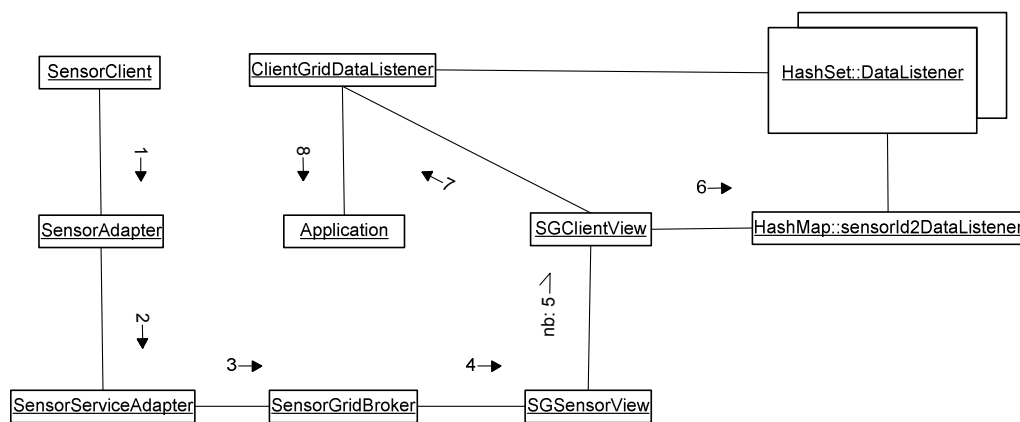


Figure 3-31 Message flow from deployed sensors to applications in a sensor grid

- 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 has subscribed to this NB stream, locates all registered ClientGridDataListeners (Sensor Data Listener) which has subscribed to data from this sensor
- 7 For each ClientGridDataListener found, notifies it for data arrival by invoking handleSensorData()
- 8 Notifies the application for data arrival

3.4.4.6 Subscribing Sensor Data

Applications that implement the SCGMMS API could receive relevant live sensor streams in the sensor grid by subscribing to them. The message flow of an application subscribes to live stream of a deployed sensor is shown below in Figure 3-32.

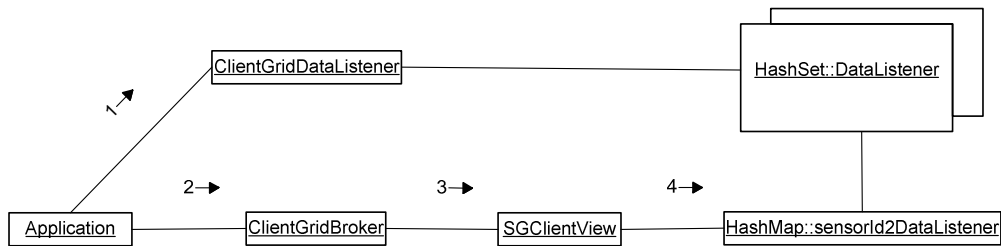


Figure 3-32 Message flow from a sensor grid to a subscribing application

- 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

3.4.4.7 Setting a Filter

The design of SCGMMS 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

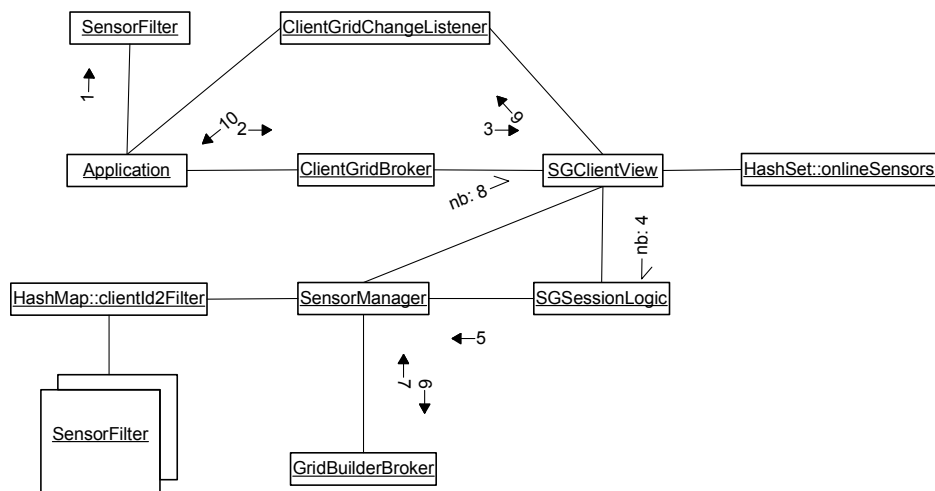


Figure 3-33 Message flow of filter setup in a sensor grid

- 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 Sensor Grid through NB, together with the `SensorFilter` object
- 5 Pass the `SensorFilter` object to `SensorManager`
- 6 Send a request to Grid Builder 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 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

3.4.4.8 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 3-34.

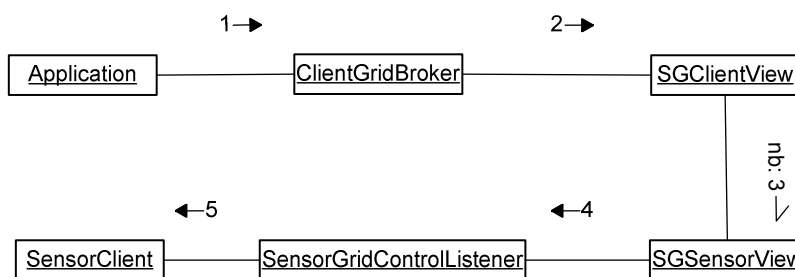


Figure 3-34 Message flow of control messages from applications to sensors in a sensor grid

- 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

3.4.4.9 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 below shows the message flow of disconnecting a sensor this way.

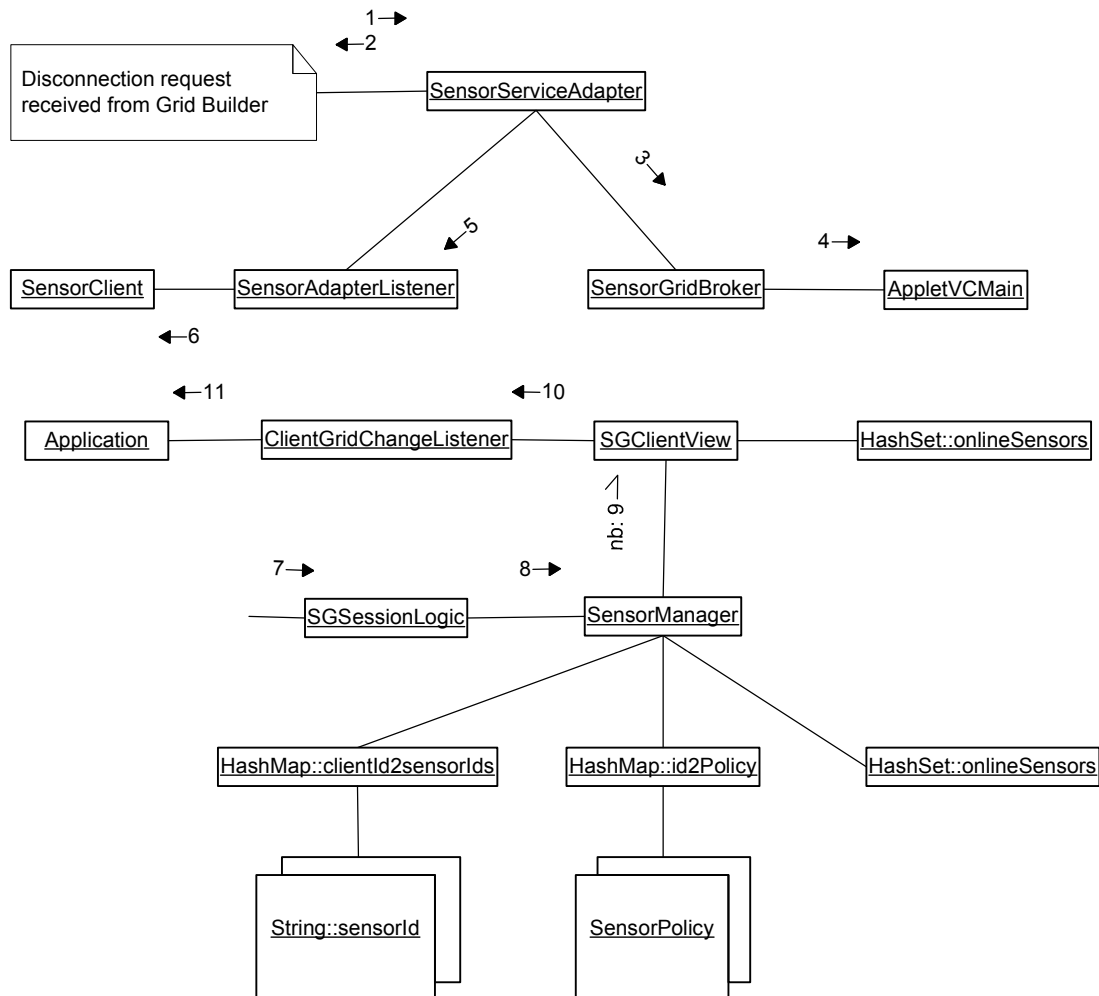


Figure 3-35 Message flow when disconnecting a deployed sensor from a sensor grid

- 1 A disconnection request is received from Grid Builder (please refer to session 3.3.5.2 for details). 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, 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
- 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

3.5 SCGMMS Application Program Interface (API)

The SCCGMMS Application Program Interface (API) allows any third party application to connect and utilize functions provided by SCGMMS. An application can do the following through the SCGMMS API:

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

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

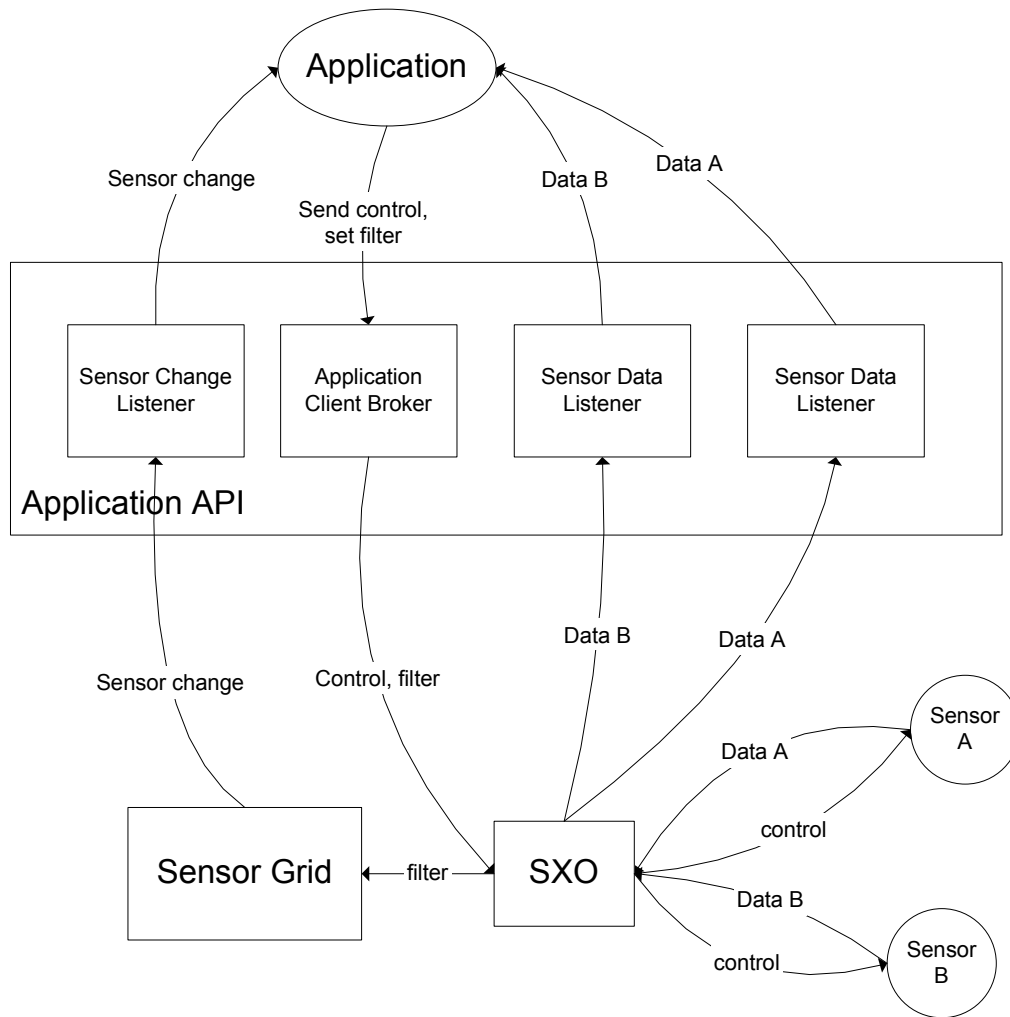


Figure 3-36 SCGMMS Application Programming Interface

Sensor Service Abstraction Layer (SSAL)

3.5.1 Overall Sensor Service Abstraction Layer Architecture

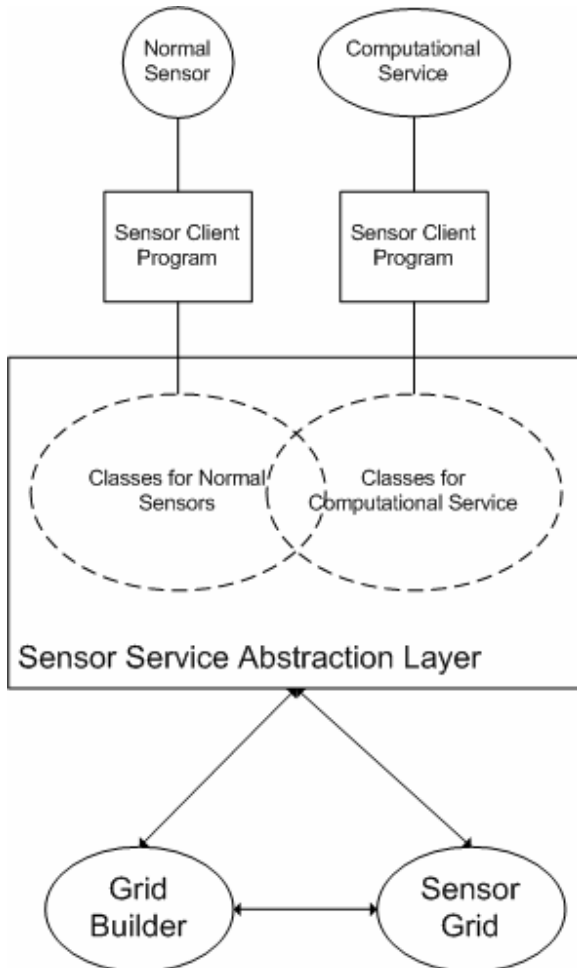


Figure 3-37 A high-level architecture of the Sensor Service Abstraction Layer (SSAL)

Sensor Service Abstraction Layer (SSAL) provides a common interface for all kinds of sensors. Sensor developers add new sensors to SCGMMS by writing **Sensor Client Programs (SCP)** which connects to SCGMMS through libraries in 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 SCGMMS.

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

Functionalities of 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 functionalities.

3.5.2 SSAL Architecture for General Sensor Services

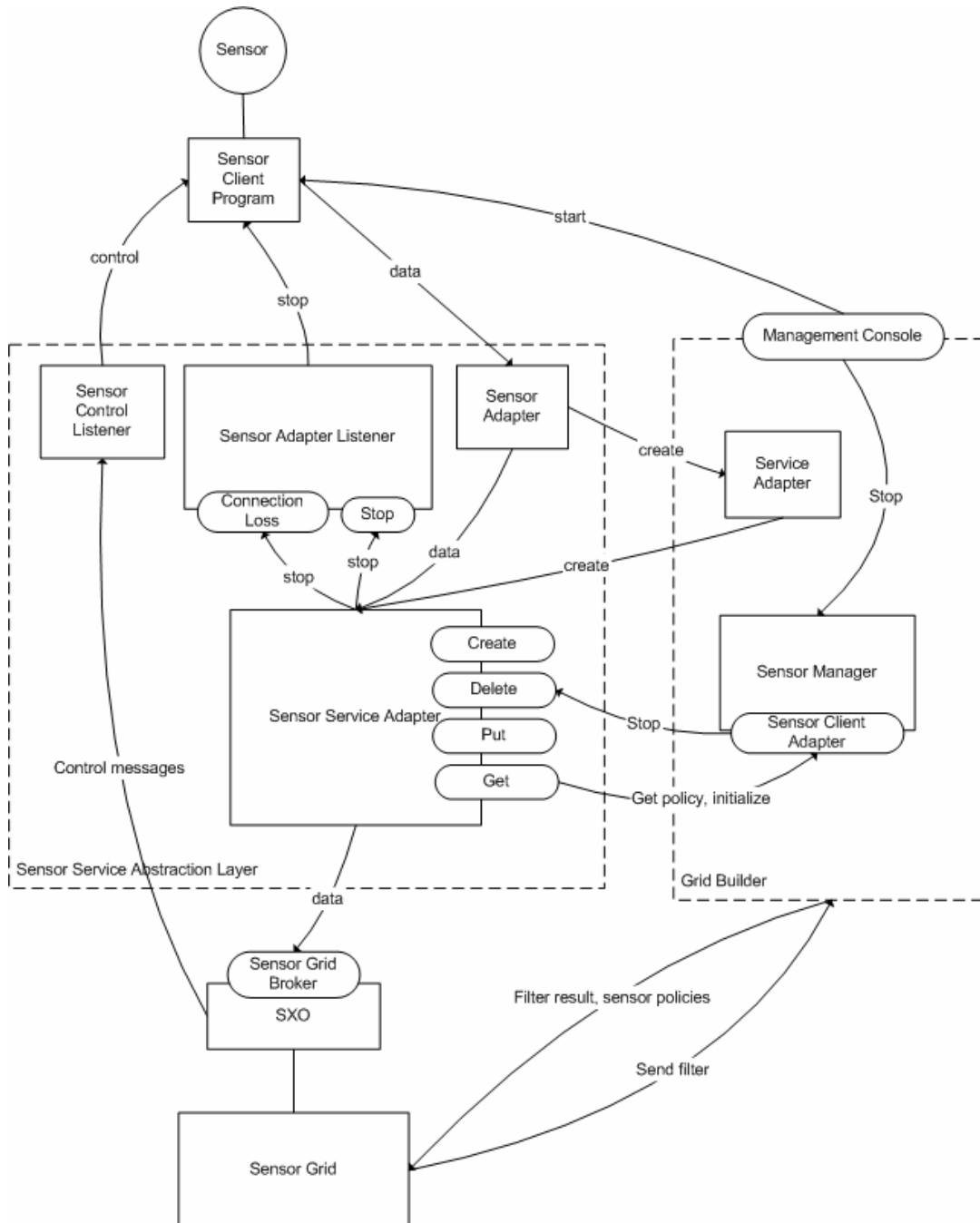


Figure 3-38 A detailed SSAL architecture for general sensor services

Figure 3-38 shows the architecture of SSAL for general sensors to be wrapped and deployed as sensor services. The following subsections explain the message flow for some basic operations.

3.5.2.1 Sensor Deployment

To deploy a sensor, the corresponding SCP has to instantiate a Sensor Adapter which notifies SCGMMS for 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 way decided by the sensor developer (e.g. run a .bat script), or it can be embedded in SCGMMS so that it can be started by GB's Management Console. For a more detailed message flow, please refer to section 3.3.5 .

3.5.2.2 Data Publishing

SCP is responsible for collecting data from the sensor, and then publishes it through Sensor Adapter. Sensor Adapter in turn forwards the data to the corresponding Sensor Service Adapter, and finally to all applications that have subscribed to its data through SXO. For a more detailed message flow, please refer to section 3.4.4.5 .

3.5.2.3 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 Sensor Adapter Listener. Currently, there is only one action supported by SCGMMS – terminating the SCP.

3.5.3 SSAL Architecture for Computation as a Sensor Service

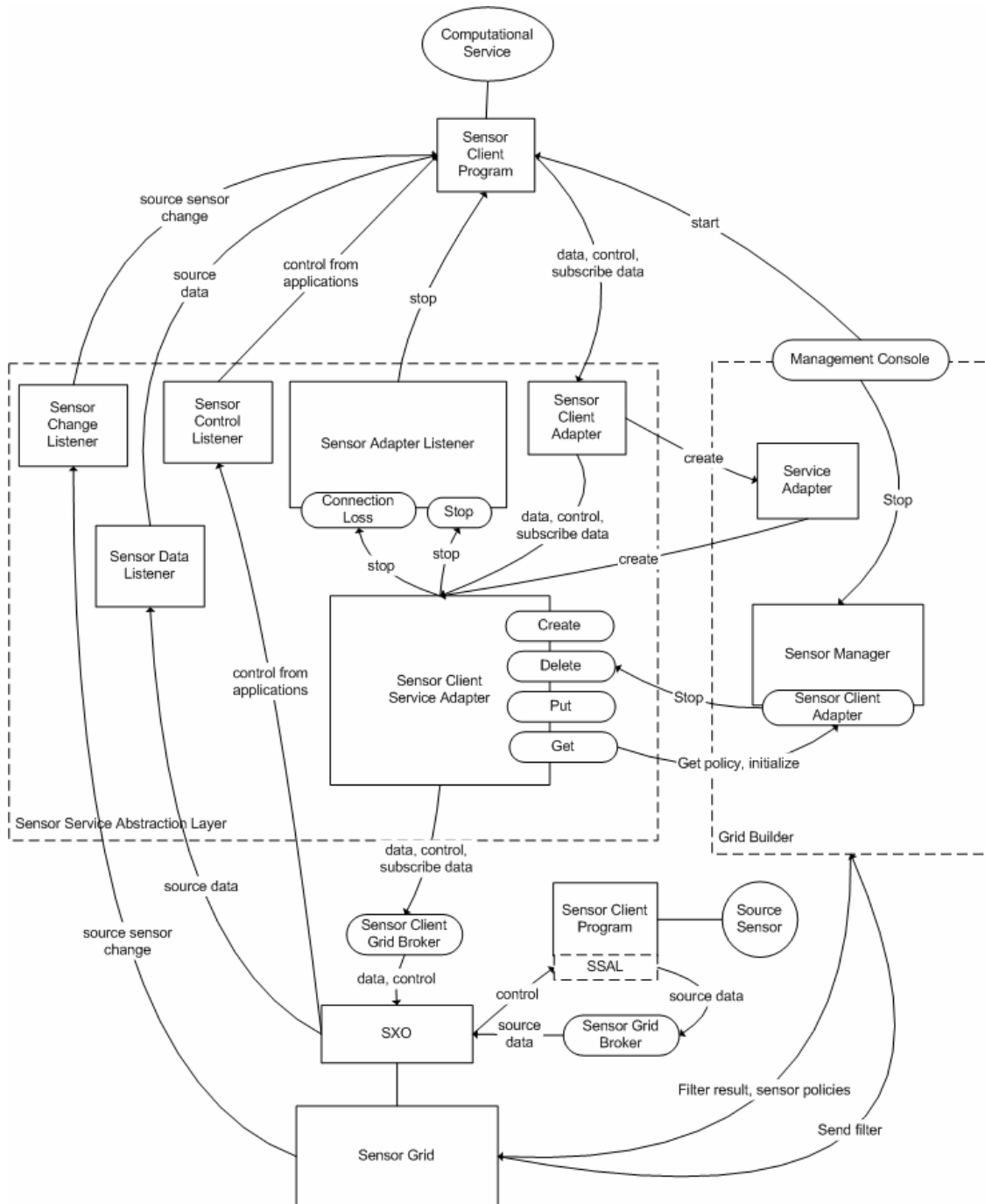


Figure 3-39 A detailed SSAL architecture for computation as a sensor service

Architecturally SSAL for Computational Service combines SSAL for normal sensors and SCGMMS API since it needs functionalities from both sides. Figure 3-39 shows SSAL for Computational Service. You can observe that components of the SCGMMS 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.

3.5.3.1 Sensor Deployment

To deploy a Computational Service, the corresponding SCP has to instantiate a Sensor Client Adapter which notifies SCGMMS for its presence and for 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.

3.5.3.2 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 through SXO. 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.

4 Advanced User-Defined Operational Pictures

Besides providing a uniform integration framework for sensors, services, grids, and grid of grids by treating or wrapping everything as a sensor service, the SCGMMS design also supports developing and deploying sensor-centric User-Defined Operational Pictures (UDOP) applications for situational awareness.

4.1 UDOP Overview

This section provides a general description of basic components in UDOP and its importance in real life application.

4.1.1 Definitions

The purpose of UDOP is to enable a user to easily choose, create, visualize and share decision-focused views of the operational environment for decision-makers to support accurate situation awareness and timely decision making in a distributed net-centric Command and Control (C2) environment. Operational environment refers to the environment where stakeholders of an operation reside. For example, the operational environment of a stock holder is the entire stock market, where the operational environment of a taxi driver is the roads, highways, tunnels etc. of the region he/she is working.

In order to make accurate decisions in an operational environment, we have to create different operational pictures which give us situation awareness in the operational environment. Developing situation awareness is a timely and expensive task which involves massive amount of information collection and data analysis. The information collected has to be decomposed, analyzed and exploited to produce useful operational pictures.

UDOP allows the user to select what information to be included or excluded from the operational picture using different filtering criteria. The filtering criteria are defined based on user's need and particular interest in the world. The selected information can then be presented, visualized and shared among all stakeholders or the operational picture using added value information products.

4.1.2 Why UDOP is Needed

UDOP is a kind of Information Management. Imagine the large amount of information in the world. In different situations we only have to utilize a probably small subset of this information. UDOP provides a systematic way for us to extract the information and let us be more aware of the current situation and able to make accurate decisions based on the extracted information.

4.1.3 UDOP Architecture

In our definition, the operational environment is composed of sensors. A sensor is a time-dependent stream of information which has a geo-spatial location. This definition is broad enough to cover most of the information sources in the world. Our proposed UDOP architecture is shown in Figure 4-1. It consists of 4 layers:

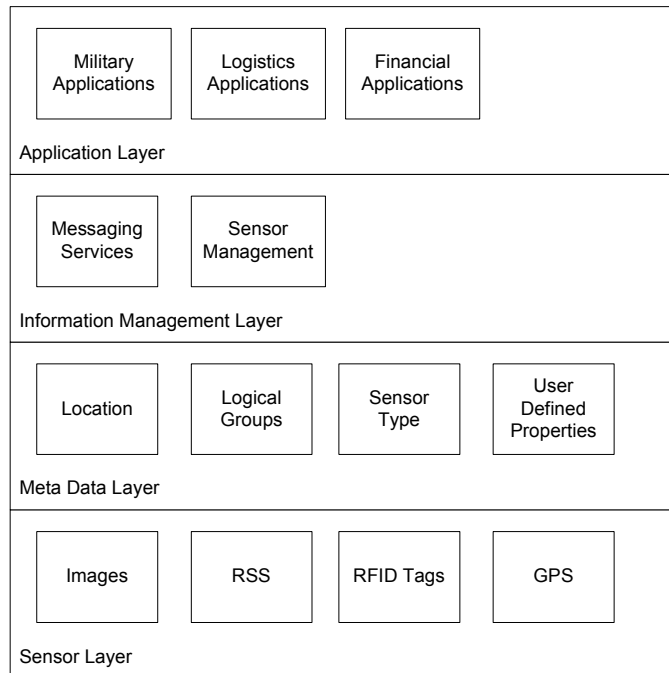


Figure 4-1 UDOP Architecture

4.1.3.1 Sensor Layer

The operational environment is composed of sensors. Sensors provide raw data which is captured dynamically from different environments.

4.1.3.2 Meta Data Layer

The Meta Data layer contains meta-data which describes the properties of sensors, which gives meaning to raw data collected from different sensors. Meta-data makes information filtering and decision-making possible.

4.1.3.3 Information Management Layer

This layer is responsible for transporting messages from sensors to applications. It contains messaging facilities which support multi-protocol and net-centric communications among sensors and applications. It also provides facilities for sensor management, such as deploying and disconnecting the sensors.

4.1.3.4 Application Layer

Application layer contains UDOP applications specialized for different operational pictures. They select, present and share information obtained from lower layers.

4.2 The Role of SCGMMS in UDOP

SCGMMS resides in the second and third layer of our proposed UDOP architecture. It is a middleware which is responsible for collecting data from all sensors in an operational environment, and provides applications with sufficient information on connected sensors for creating different operational pictures using UDOP approach. There are two types of potential users of SCGMMS:

Application Developers

Application developers are domain experts who develop UDOP applications specific to their domains. SCGMMS provides an Application API which allows developers to retrieve sensor data and meta-data from the sensors.

Sensor Developers

Sensor developers are information providers who want to make available some raw information for different applications to use. They have to connect their sensors to SCGMMS through Sensor Service Abstraction Layer (SSAL). SSAL requires sensor developers to define the meta-data of their sensors so that applications can do filtering according to this meta-data.

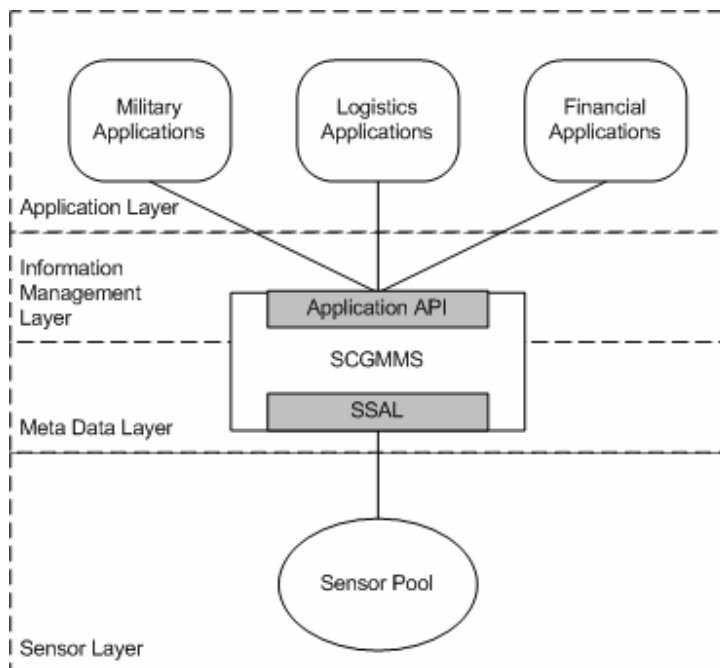


Figure 4-2 Role of SCGMMS

4.2.1 How SCGMMS supports UDOP Development

Several generic core pieces of functionality are needed to enable the UDOP capability. They include:

1. **Data access** mechanisms for building a UDOP from the outputs of systems of record using net-centric means
2. **Visualization and presentation** mechanisms to provide the requisite historical, current, and anticipatory situation awareness
3. **Data selection and filtering** to create derived, added-value information products from the raw data inputs and displayed data and to extract insight based on the content therein
4. **Sharing and collaboration** tools to enable shared situation awareness and collaborative decision-making based on the decision-focused view created through the UDOP

SCGMMS supports the development of UDOP applications by providing data access and selection functionalities which meet UDOP requirements. Applications which utilize SCGMMS shall be able to achieve visualization and collaboration with minimum effort. The following sections describe how these functionalities are supported.

4.2.1.1 Data Access

One of the requirements of UDOP is to employ a net-centric, loosely coupled and standards-based data access mechanism. Data access of SCGMMS fulfills all the requirements. Logically SCGMMS provides a centralized, perpetual session for monitoring and collecting information from all sensors, at the same time allows applications to collect information according to their UDOP requirements. Physically all applications and sensors are connected by a distributed brokering network which supports messaging with various protocols through the internet. Therefore, both sensors and applications can reside in anywhere in the world where internet connection is available. Detailed description of message flow between SCGMMS, sensors and applications can be found in section 3.4.1

In terms of standard compliance, the communication between SCGMMS and sensors is SOAP compatible. Although the current implementation does not take into account of external communication with SCBGM through Web Service Interface, it will be supported in near future.

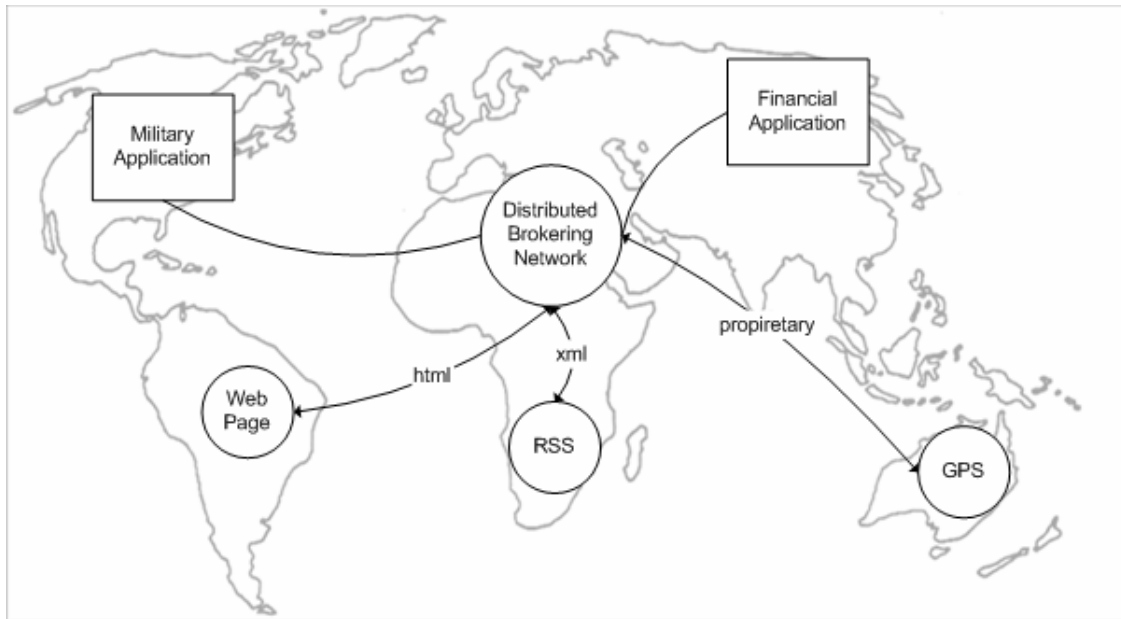


Figure 4-3 Distributed Architecture for Data Access

Data Model

Most of the data are acquired through an event-driven model. Sensors in different geo-spatial locations continuously publish data into the distributed brokering network. SCGMMS helps routing the data to all connected applications according to their UDOP requirements. Applications are notified for each data arrival through data listeners.

Some sensors are capable of receiving requests from applications and perform some actions in return. These actions are sensor-specific. Some of them even take data from other sensors and output processed data as response.

Quality of Data

Quality of a data is essential for decision-makers to know the limitations in their knowledge of some operational situation. It can be measured with the following aspects:

Correctness

Currently there is limited verification on the correctness of data received from sensors. The collected data is just forwarded to applications directly on its arrival. However, SSAL requires each sensor's property and its data format to be well defined. This guarantees the correctness of data. However, malicious activities such as impersonation can not be detected in the current implementation.

Consistency

In terms of consistency, SSAL requires each piece of data to be annotated with a timestamp. The timestamp can be used by applications to check for consistency and timeliness of data. (e.g. discarding out-of-date data, checking order and frequency of data arrival).

4.2.1.2 Data Selection and Filtering

Each UDOP application is only interested in its own domain-related information extracted from the large raw data pool supported by SCGMMS. SCGMMS should provide a flexible and efficient data selection scheme which can possibly support users from very different domains. There are two main features in SCGMMS which makes data selection flexible and efficient.

Classification

Classification is a set of meta-data associated with each sensor describing the property of the sensor and its data. Classification serves the following functions:

5. Allows sensor developers to describe the characteristics of sensors and what service the sensor provides
6. Allow each sensor to be identifiable uniquely
7. Allow sensors with similar properties to be logically grouped together
8. Allow application to differentiate different sensors
9. Allow meaningful visualization of sensor data at application side

Only with this meta-data decision makers are able to decide which sensor is under his/her interest. SCGMMS requires every sensor to have a common set of properties. They include:

Property	Description
Sensor ID	Each sensor has a unique ID for identification
Group ID	Sensors can be assigned to different logical groups for easier management. Group ID identifies the group
Sensor Type	Each sensor has a unique type
location	Geo-spatial location of the sensor, including street, city, state/province and country
historical	Defines whether different pieces of data from the sensor are temporally related to one another for give meaning
Sensor Control	Sensors are able to receive and reacts to control messages
User Defined Property	Some properties which are specific to the sensor are defined here

As the number of supported sensors grows, the classification scheme should be extended to support more generic and extensive description on the properties of sensors.

Filtering

Filtering refers to the action of selecting sensors based on their properties defined in the classification. Currently SCGMMS supports the application of Union and Intersection set operations on each of the sensor property defined in the previous section. Filtering follows the request/response model where an application defines a “filter” which contains set operations on different fields defined in the classification. The filter is then sent to SCGMMS as request. In response SCGMMS replies the application with a set of sensors which match the filter. The application can then subscribe to data of these sensors through the SCGMMS API.

For example, if a decision-maker wants to locate all GPS and RFID sensors in US or UK, the corresponding query looks like:

**sensorType=GPS \cap sensorType=RFID \cap location=US \cup
sensorType=GPS \cap sensorType=RFID \cap location=UK**

The filtering format is defined by the Application API. It is the application's responsibility to visualize the input of these set operations using GUI for user friendliness. Here is a sample GUI for the query above:

Sensor Filter

Sensor ID:

Group ID:

Sensor Type:
 ▼

City:

AND **OR**

Filter to Apply:

sensorType=GPS
 [AND] sensorType=RFID
 [AND] city=US
 [OR] sensorType=GPS
 [AND] sensorType=RFID
 [AND] city=Hong Kong

Apply Filter **Reset**

Figure 4-4 A sample GUI-based sensor selection filter

4.2.1.3 Visualization and Presentation

Visualization and presentation are application specific. The filtering mechanism of SCGMMS is flexible enough to allow applications to visualize and present data according to their own UDOP. The classification scheme also provides meta-data so that data can be visualized in meaningful ways.

4.2.1.4 UDOP Management

Given that data access is in a net-centric and distributed manner, applications in different parts of the world should be able to retrieve data from sensors in a consistent manner.

The applications itself can further utilize this advantage to deploy operational picture specific collaboration capabilities. In the next section, we will introduce an application which is built on top of SCGMMS supporting sophisticated sharing and collaboration.

To allow a flexible way to manage UDOPs, SCGMMS should provide a UDOP Management Service for creating, modifying, reusing, storing and sharing UDOPs. SCGMMS allows such service to be created at the Sensor Layer of our UDOP architecture by allowing a sensor to receive and publish data from and to a specific application.

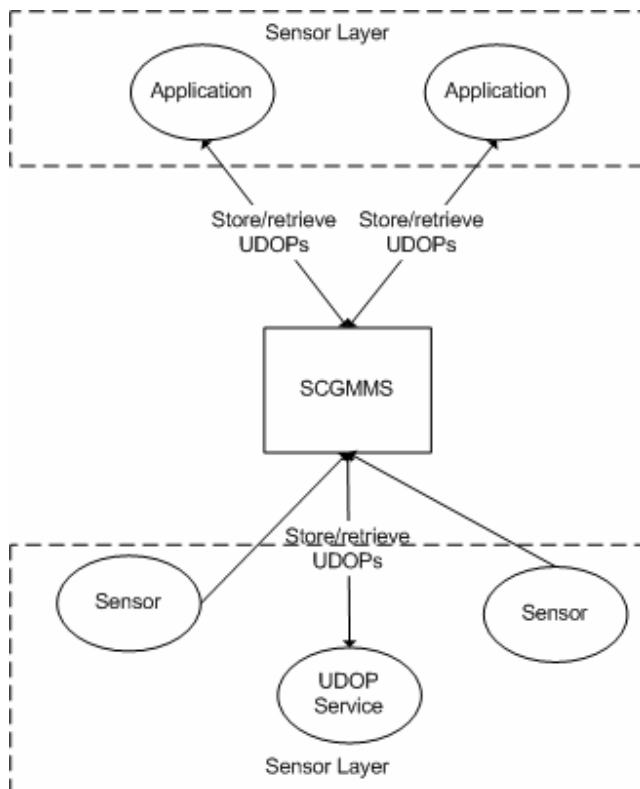


Figure 4-5 UDOP Management

Figure 4-5 shows the overall architecture of application storing and retrieving UDOPs from a UDOP Service through SCGMMS. The exact architecture and implementation of UDOP Service will be discussed in section 4.3.

4.3 UDOP Service

UDOP Service is a UDOP management solution provided by SCGMMS and Sensor Sharedlet together to illustrate how SCGMMS can support the development of UDOP applications.

4.3.1 Overview

In Sensor Sharedlet, different operating pictures can be created by:

1. Drag-and-drop sensors to the presentation area
2. Applying different filters

The goal of UDOP Service is to provide management service for these operating pictures, including creation, modification, removal and storage. We call these operating pictures “UDOP Templates”.

4.3.1.1 UDOP Template

Within each UDOP Template, the following information will be stored as attributes of the Template:

UDOP ID

This is a label for identifying the UDOP Template.

UDOP Description

The description of the UDOP Template includes information such as general description of the Template and what situation awareness this Template gives. The description will be used for Template searching.

Presentation Info

This includes any presentation specific information such as which sensors are being displayed in the presentation area of Sensor Sharedlet (i.e. the 4 panels), and in what order they are being displayed.

Filtering Criteria

This includes the filtering criteria applied to the sensor metadata.

Sensor Description

The user is allowed to write his/her own description to each of the sensors based on his/her UDOP requirements.

A UDOP Template can be saved, loaded, modified and deleted. Each time the Template is modified, the old Template will be saved as history for later retrieval.

4.3.2 Architecture

To cope with the architecture of SCGMMS, a UDOP Service is implemented as a sensor. It has to be deployed in one of the GB domains just like other sensors and communicates with SCGMMS through SSAL.

A single UDOP Service will be responsible for managing a number of UDOP Templates. Each UDOP Template will be stored in internal data structure such as classes and hash tables within the UDOP service.

UDOP Templates will be created in the Sensor Sharedlet, which will be saved/retrieved to/from the UDOP service through sending control messages and receiving sensor data. The overall architecture is illustrated in Figure 4-6.

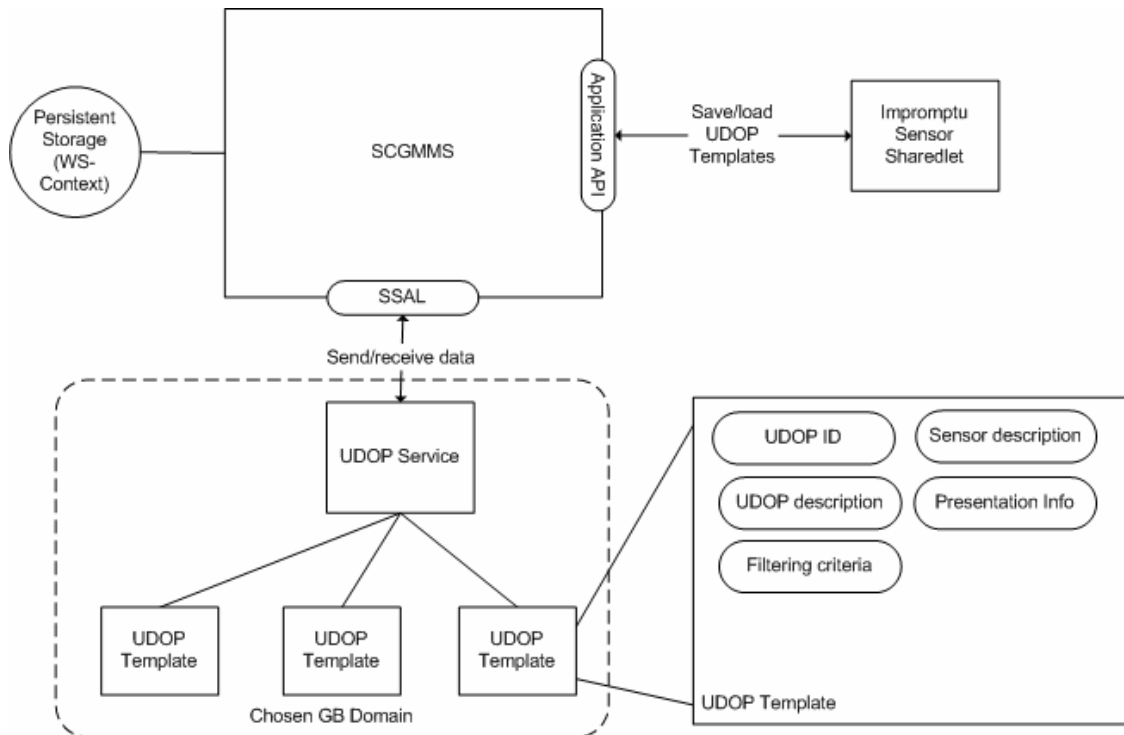


Figure 4-6 UDOP Service Architecture

4.3.3 Detailed Description

In this section, the internal data-flow of performing various UDOP management actions on UDOP Service will be described in detail.

In the system standpoint, the UDOP Service is just a sensor communicating with SCGMMS through SSAL. Performing various management actions is equivalent to sending control messages to the UDOP service. UDOP services replies by sending data back to the application clients. Figure 4-7 shows the general idea of UDOP management.

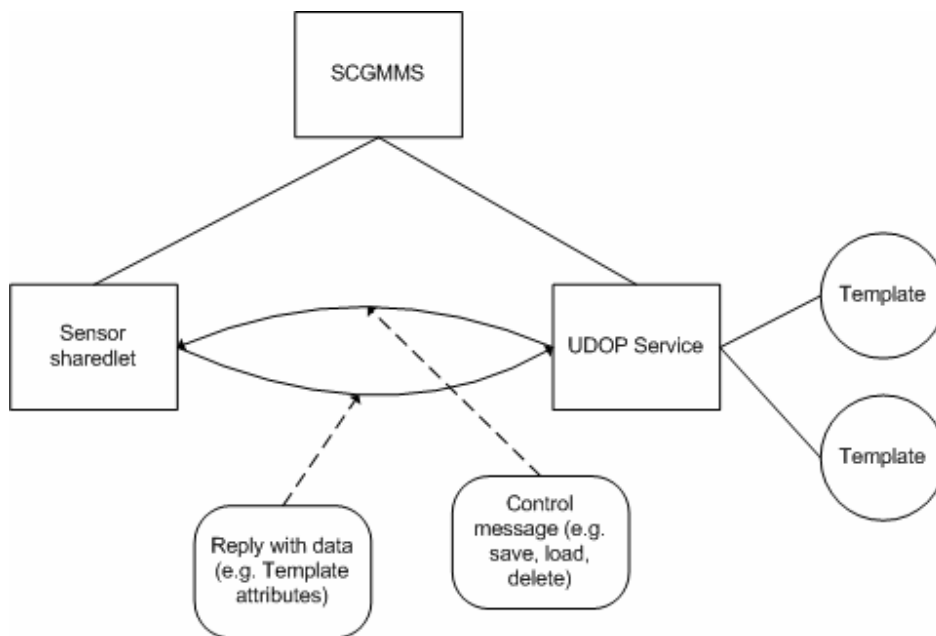


Figure 4-7 System view of UDOP Template management

4.3.3.1 Saving a UDOP Template

Figure 4-8 below illustrates the dataflow of saving a UDOP Template to UDOP Service.

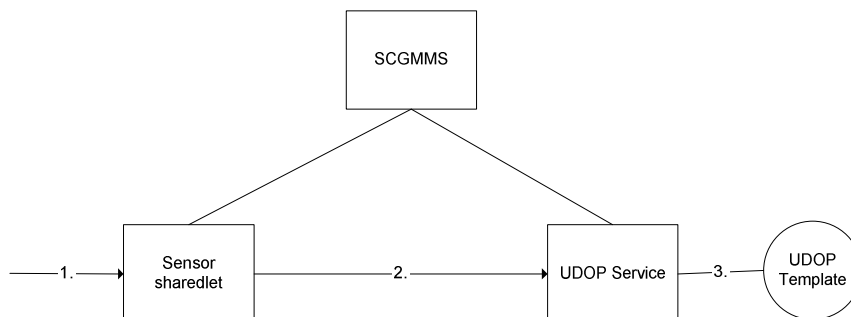


Figure 4-8 Dataflow of saving a UDOP template to UDOP service

1. The user created an operating picture in Sensor Sharedlet and decides to save it as a UDOP Template. A “save” request is generated from the GUI.
2. The Sensor Sharedlet, which is already connected to a UDOP Service through Application API, sends a control message with type “save” and corresponding details of the UDOP Template as parameters of the message to the UDOP Service.
3. UDOP Service receives the message and registers the new UDOP Template within the Sensor Client Program

4.3.3.2 Retrieving a list of UDOP Templates

In Figure 4-9 we illustrate the dataflow of retrieving a list of UDOP Templates from the UDOP Service. This process is automatic – meaning that the Sharedlet automatically update the list by sending a query to UDOP Service for every 5 seconds interval. Note that the UDOP Service only replies Client A (the application which initiates the request) with the UDOP Template list instead of broadcasting the response to all clients.

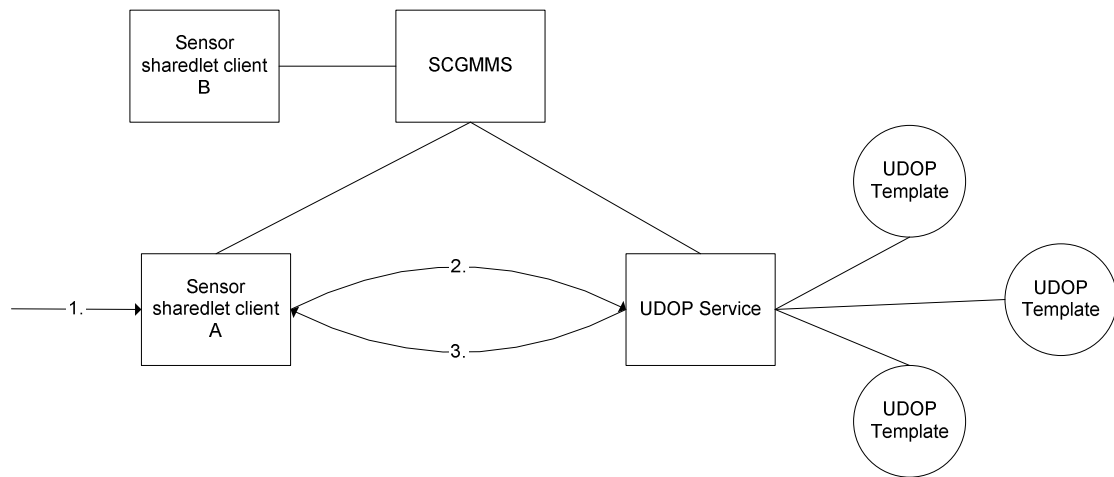


Figure 4-9 Dataflow of retrieving a UDOP template from UDOP service

1. Sensor Sharedlet generates a query request to the UDOP Service for every 5 seconds
2. A control message with type “get_list” is sent to the UDOP Service
3. UDOP Service replies with data of data type “UDOPData” to sharedlet client A. The Sharedlet then update its GUI

4.3.3.3 Opening a UDOP Template

After the Sensor Sharedlet has retrieved a list of UDOP Templates, the reply message actually contains details of each Template. To open a Template, the Sharedlet simply read the retrieved data and load GUI according to Template data.

5 An Advanced Technology Demonstrations of SCGMMS

5.1 Demonstrations of SCGMMS

Two technology demonstrations during the course of SCGMMS development took place in international symposium and conferences. The first globally deployed live demonstration of an early SCGMMS prototype was in Supercomputing 2007 in Reno, Nevada in November 2007. Another advanced technology demonstration of a further developed SCGMMS prototype was deployed globally in the International Symposium on Collaborative Technologies and Systems in Irvine, California in May 2008.

The demonstration scenarios are discussed here.

5.1.1 Demonstration of SCGMMS in Supercomputing 2007

The first demonstration was an early version of SCGMMS. . A key illustration in this demonstration was the integration and real-time sharing of video and GPS sensors streams globally in a message-based collaboration application called Anaba Impromptu. The objective was to support easy organization, presentation and visualizataion of live video and GPS streams to enable rapid situational awareness.

The demonstration operation environment comprised of four locations, which included Reno (Nevada), where the Supercomputing Conference took place, San Francisco (California), Bloomington (Indiana University) and Hong Kong International Airport. In this demonstration, a 4-location collaborative session in which simultaneous, live streams of video and GPS sensor data were streamed, multicasted and synchronously shared among participants of the session, allowing each participant to present her/his own video and GPS streams while visualizing data from all live video and GPS in real-time and a collaborative manner. The sensors employed in the demonstration were:

1. **Nokia N800:** An Internet Tablet PC which has considerable computing power with supports for both Wi-Fi and Bluetooth connections. It is also equipped with a camera capable of capturing and publishing live H.263 video stream. Four Nokia N800 were used.
2. **GPS sensor:** It is a portable GPS receiver, which receives geospatial location information (e.g., latitude, longitude, etc) from satellites. Four GPS sensors were used.

Table 5-1 A sample sensor types and attributes table used in Supercomputing 2007 demonstration

<i>Sensor Type</i>	<i>Attributes</i>
GPS	<ul style="list-style-type: none">- Time- Latitude- Longitude- ID
Video/Audio	<ul style="list-style-type: none">- Video stream- Audio stream

The base application used for the demonstration is the Anabaz Impromptu collaboration client which supports synchronous sharing of Webcam streams for video conferencing, Voice over IP for audio conferencing, whiteboard, Web browsing and any Windows applications, among other features. The video conferencing collaborative application module called the Video sharedlet in the Impromptu client was customized to support the Nokia N800's H.263 video format. A "Map Demo" sharedlet was implemented for presentation and visualization of live, streaming GPS data. Figure 5-1 illustrates when the "Map Demo" was enabled, a drop-down menu was available for selecting which GPS stream was to be presented. In this case, "Show All" was chosen and the four GPS streams corresponding to the locations of the four video sensors (the Nokia N800 built-in Webcams) from Reno (Nevada), San Francisco (California), Bloomington (Indiana) and Hong Kong International Airport were displayed in an integrated Google map. The Google map was shared synchronously among all participants in the collaborative session.

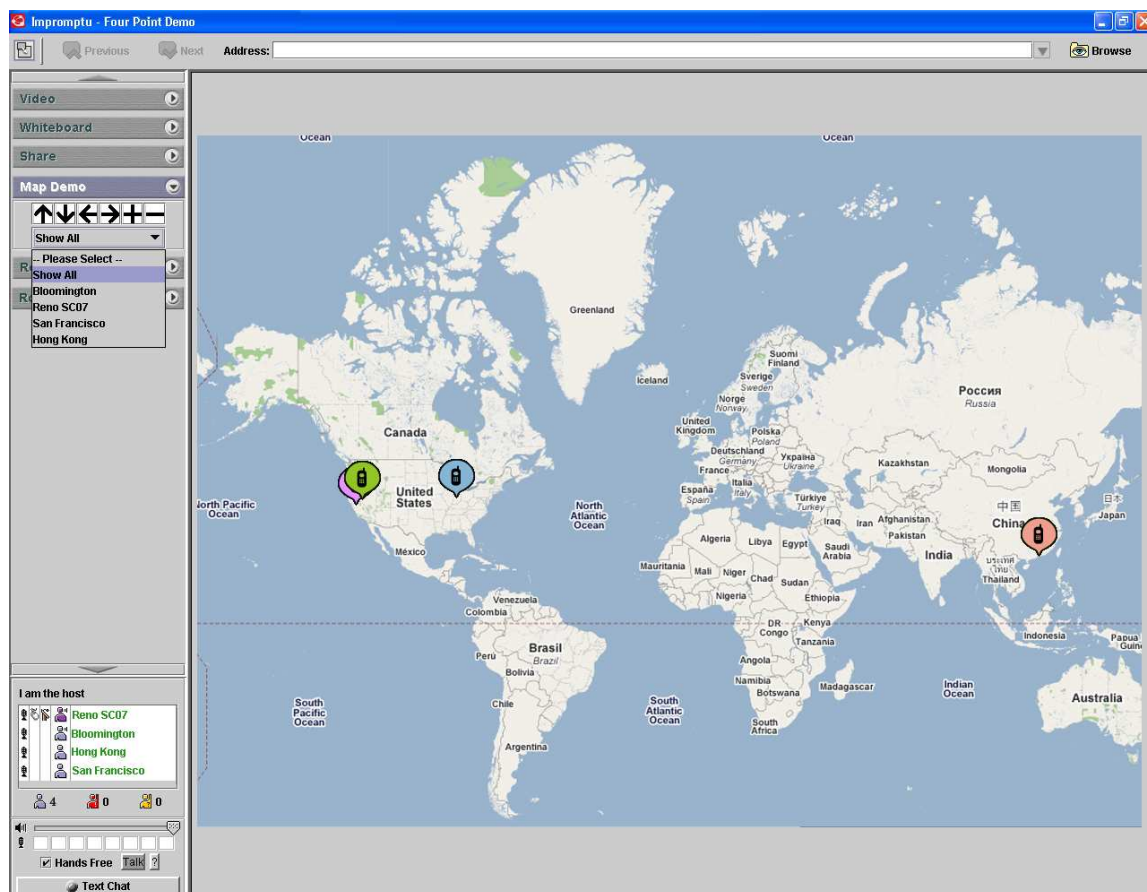


Figure 5-1 A map sharedlet shows live GPS streams of sensor locations from all over the world

Figure 5-2, 5-3, 5-4 and 5-5 illustrate the visualization of individually selected GPS stream for display in Impromptu Map sharedlet.

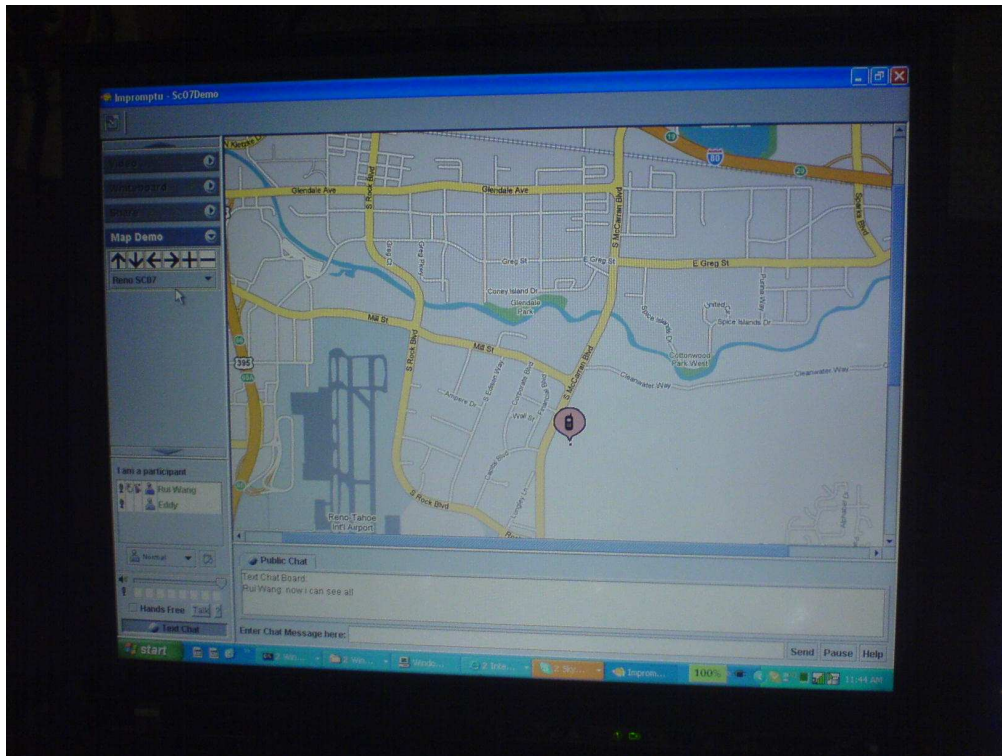


Figure 5-2 A map sharedlet shows the live GPS stream from Supercomputing 2007 (SC07) in Reno, Nevada

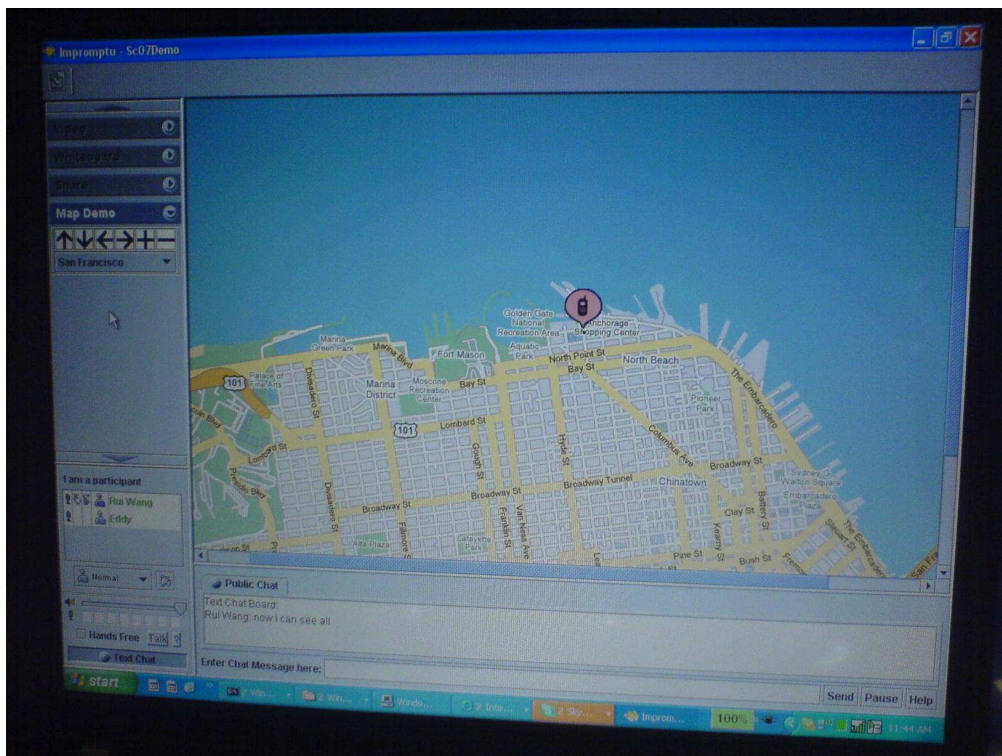


Figure 5-3 A map sharedlet shows the live GPS stream from San Francisco, California

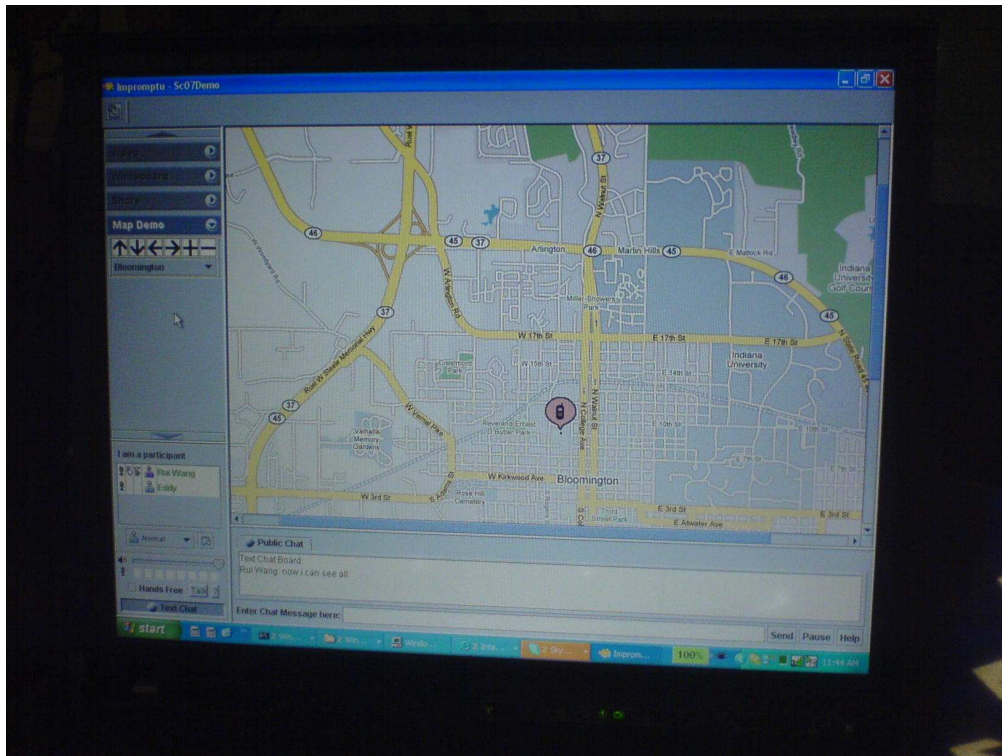


Figure 5-4 A map sharedlet shows the live stream from Bloomington, Indiana

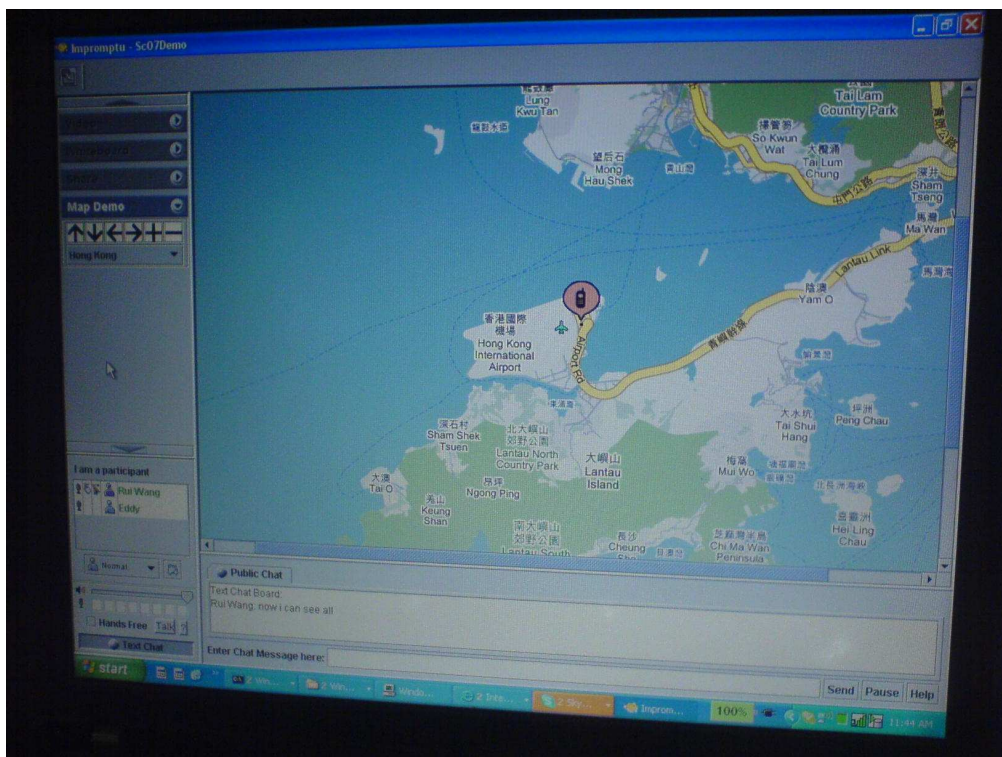


Figure 5-5 A map sharedlet shows the live GPS stream from the Hong Kong International Airport

5.1.2 Demonstration of SCGMMS in CTS 2008

A significantly improved SCGMMS prototype was demonstrated in the International Symposium on Collaborative Technologies and Systems in May 2008 in Irvine, California. The entire architecture for the Sensor-Centric Grid of Grids Middleware Management System was designed and prototyped. The two interfaces, the Sensor Service Abstract Layer (SSAL) API for sensor developers to “plug” their sensors into the sensor-centric grid of grids, and the SCGMMS API for application developers to leverage sensor streams that are traveling or queued up in the grid.

Key demonstration objectives were to illustrate the ease of use of SCGMMS to develop, deploy, management, organize, present and visualize collaborative geo-coded, sensor-centric grid applications with UDOP/COP capabilities. The demonstration scenario as depicted in seamless virtualized and integrated globally distributed sensors and other distributed resources such as modeling and simulations, or computations uniformly into a single system – a sensor-centric grid of grids.

The demonstration comprised of 3 locations – Irvine (California), Bloomington (Indiana) and Hong Kong. The sensors employed were:

- **GPS sensor:** It is a portable GPS receiver, which receives geospatial location information (e.g., latitude, longitude, etc) from satellites.
- **PC Web-camera sensor:** PC compatible web-cameras are available for capturing high quality video stream capturing.
- **RFID sensor:** The Mantis RfCode M220 reader and RfCode M100 active tags are used. The reader senses information about RFID tags such as the signal strength, motion, temper and panic and encapsulates the information in tag event messages.
- **Lego Robot:** We used the Lego Mindstorm NXT robots as sensor carriers for our application. Two types of robots were assembled. One is a humanoid called Alpha Rex and the other a vehicle called Tribot. We implemented eight environmental sensor types on the three Lego robots. The environmental sensors were Ultrasonic, Sound, Light, Touch, Gyroscope, Compass, Accelerometer and Thermistor. The Lego robots can also respond as instructed by taking programmable commands from any collaborative session participants.
- **Wii Remote sensor:** A Wii Remote is normally used as the primary controller for Nintendo’s Wii console game. A Wii Remote can detect infrared sources and determines their positions. In this case, a Wii Remote was enabled via the SSAL to become a sensor service and deployed to a sensor-centric grid. In the demonstration a Wii Remote sensor was primarily used to control any Lego robots deployed to the sensor grid, independent of the geographic locations of the respective robots. A Wii Remote sensor could also detect infrared sources and publish their relative locations to a sensor grid.
- **Video Edge Detection (VED) sensor:** VED is a video processing algorithm that performs real-time detection of object motion in video streams and highlights the motion area by drawing a minimum rectangular bounding box around it.

The demonstration scenario is depicted in Figure 5-6.

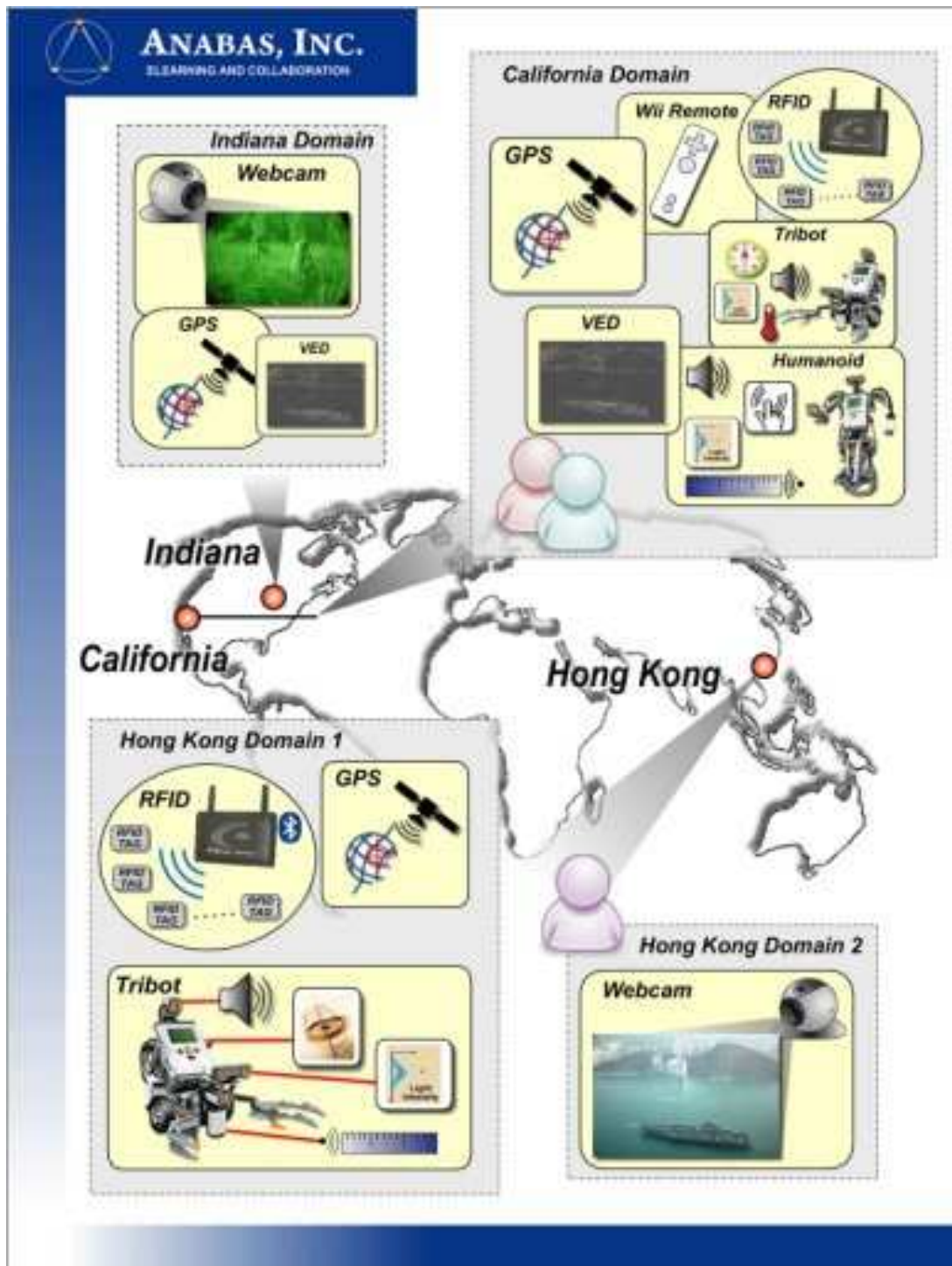


Figure 5-6 SCGMMS Multi-location, multi-robot, multi-sensor demonstration scenario

In Irvine, a GPS sensor, a Wii Remote sensor, an RFID reader sensor and multiple RFID tags, a VED sensor and 2 Lego robots – a Tribot and a Humanoid each carrying multiple Lego sensor types were deployed. In Bloomington, a GPS sensor, a VED sensor and a Webcam video sensor were deployed, while in Hong Kong a GPS sensor, an RFID reader and multiple RFID tags, a Webcam video sensor and a Lego robot – a Tribot carrying multiple Lego sensors were deployed.

Even though several sensor-bearing robots, and various physical and software sensors were deployed and geographically dispersed, all the geographically dispersed participants in an Impromptu sensor-centric collaboration session were able to collaborate, access, exchange, organize, present and visualize all sensor streaming data freely and easily as if all the sensors and participants were in the same place and on a single computer system.

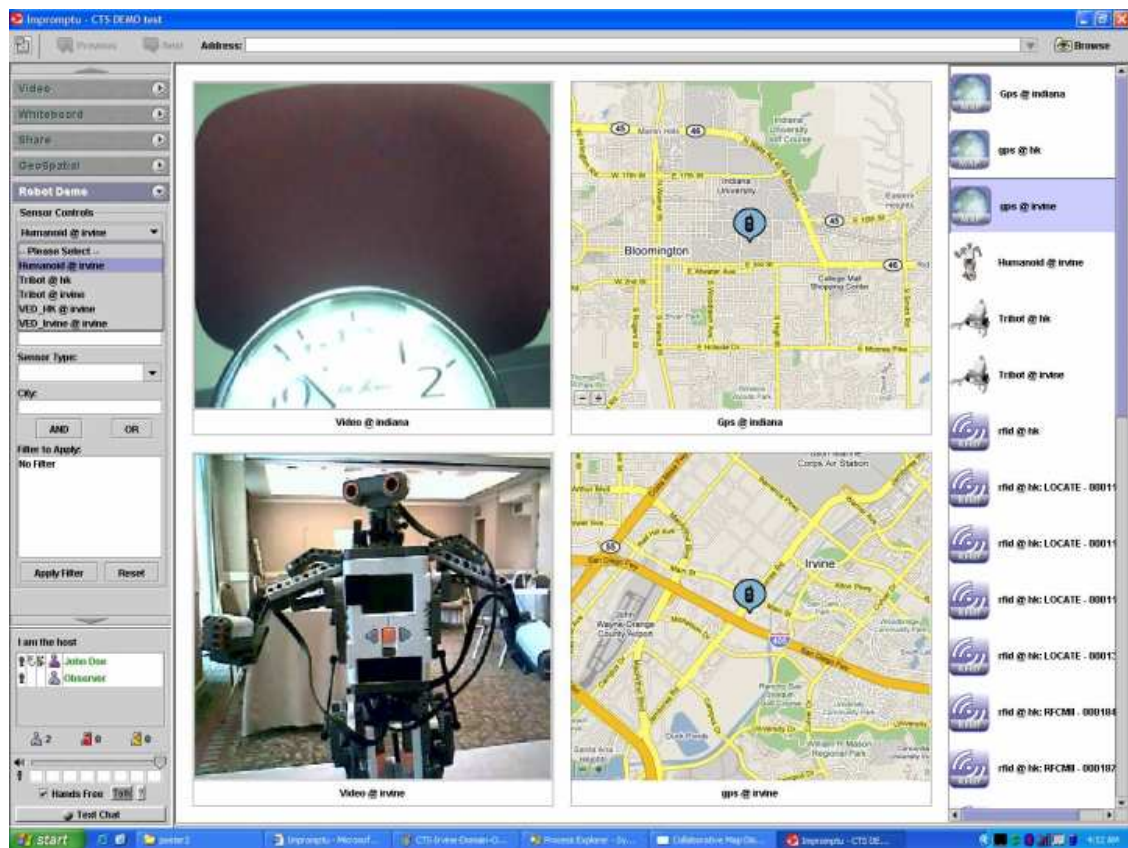


Figure 5-7 GPS and video sensor streams from Irvine and Bloomington

Figure 5-7 depicts that while many different sensor types and sensor were deployed as could be seen on the scrollable sensor list on the right hand side of the Impromptu client, a user of the system decided to dynamically create a User-Defined Operational Picture by selecting and organizing the presentation and visualization of four sensor streams – video stream from Bloomington, Indiana on the upper left, mapping of GPS sensor stream from Bloomington, Indiana on the upper right, video stream from Irvine, California (CTS 2008

site) on the lower left and mapping of GPS stream from Irvine, California on the lower right. The video stream from Bloomington showed a running clock while the video stream from Irvine showed a Lego Humanoid robot in the demonstration site.

As shown in Figure 5-8, a user of the system created another User-Defined Operational Picture on-the-fly and shared it with all other collaborative session participants, this time by selecting to organize and present the visualization of the video and GPS streams from Irvine on the upper left and right, and the video and GPS streams from Hong Kong on the lower left and right.

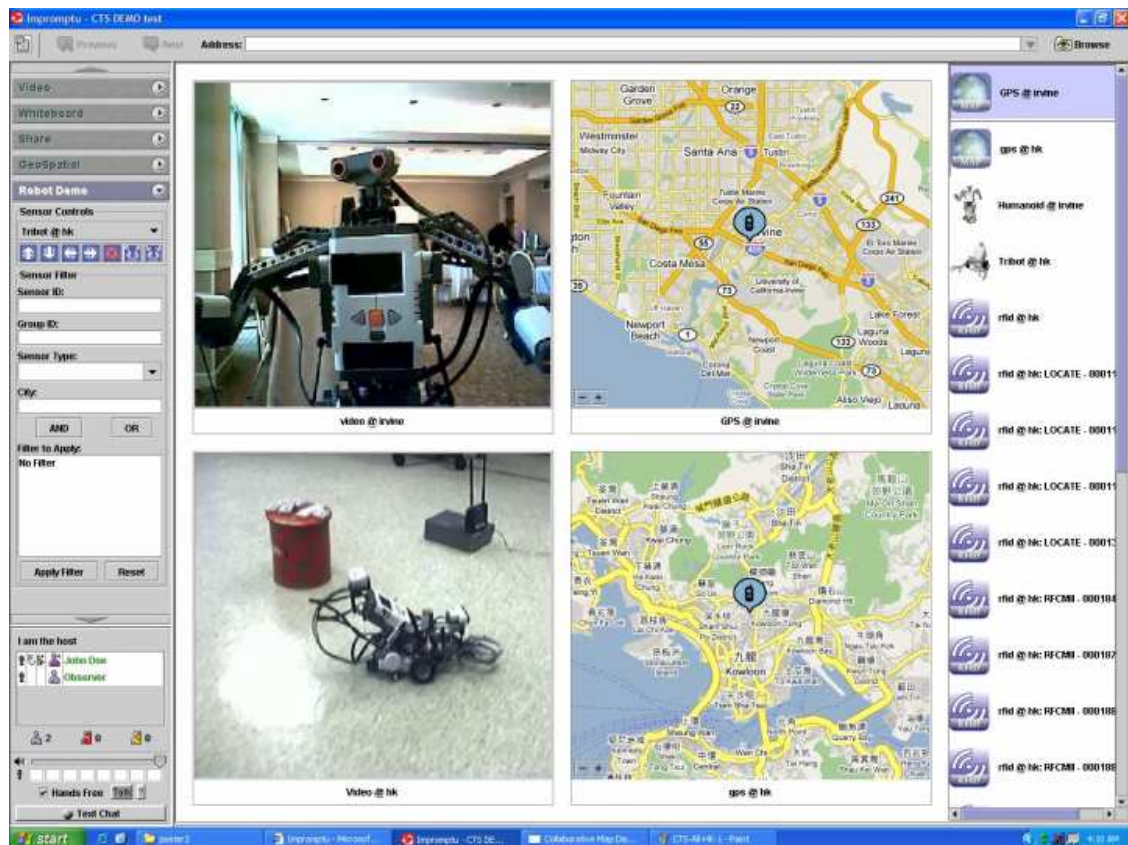


Figure 5-8 GPS and video sensor streams from Irvine and Hong Kong

A new UDOP that depicted an operational picture entirely from sensor sources from Hong Kong is shown in Figure 5-9. On the upper left, the data from the Webcam video sensor that was pointed at the operational environment of a Lego Tribot robot in Hong Kong in an area with an RFID reader and ten RFID tags placed on top of a red jar was streamed live to the sensor-centric grid and subscribed by the Impromptu client. On the upper right was the visualization of the live data from the four sensors carried by the Lego Tribot on its left. The Lego Tribot carried an Ultrasonic sensor to measure distance, Sound and Light sensors for sound and light intensity, Gyroscope sensor to measure the rate of rotation of the Tribot over time. On the lower left was the visualization of RFID

signal strength data streams from ten RFID tags in the neighborhood of the RFID reader, and on the lower right was the live GPS location stream indicated on a Google map.

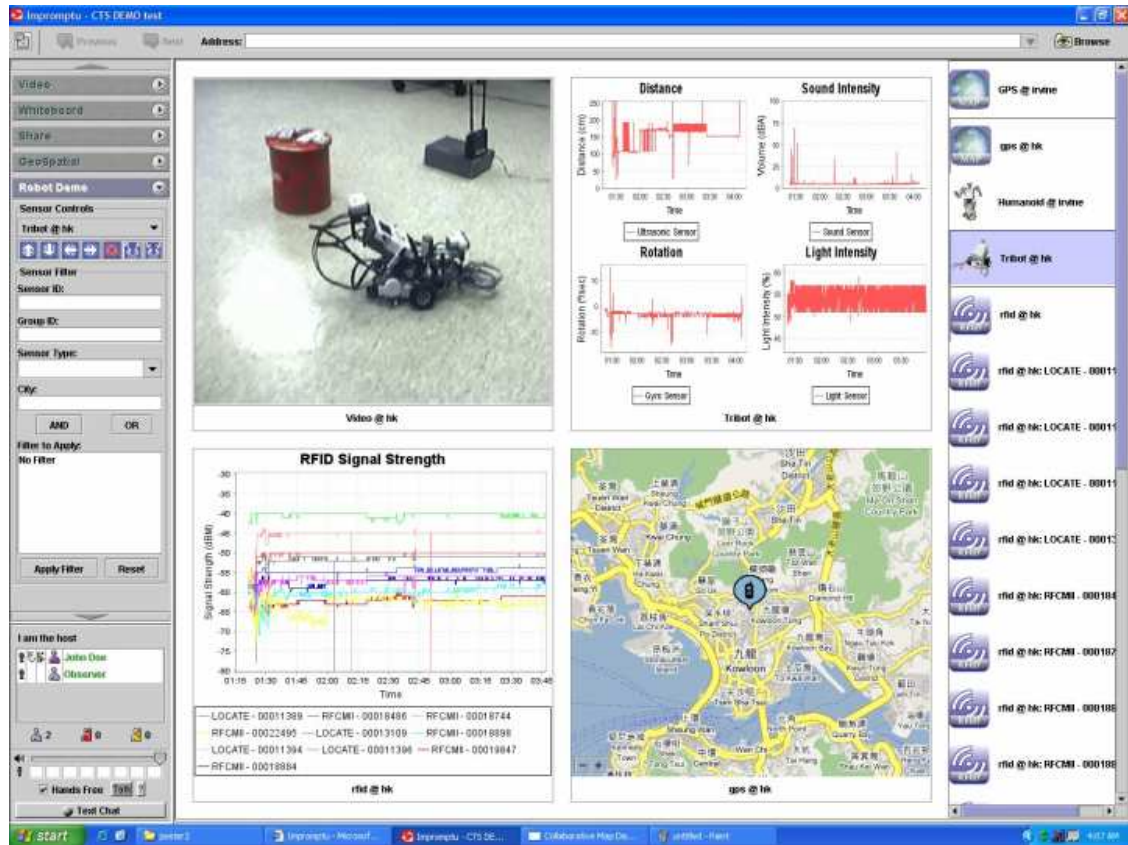


Figure 5-9 A UDOP dynamically created from remote sensor streams

In the UDOP that is shown in Figure 5-10 the visualization of the live video stream from Hong Kong was placed on the upper right. The movement of the Lego Tribot in Hong Kong was remotely controlled by the Wii Remote sensor deployed in Irvine. As the Lego Tribot was remotely controlled via the sensor-centric grid to roam around, the live video stream from the Webcam pointing at the Tribot was multicasted in real-time via the sensor-centric grid transport layer, Narada Brokering, to the Video Edge Detection (VED) sensors deployed in Irvine and Bloomington. The motion-detected, live video streams from the Irvine and Bloomington VED sensors were simultaneously shown in the lower left and right of the UDOP, respectively. The real-time updated rectangular bounding boxes represented the areas where motion was detected in live video streams.

In this UDOP, the Wii Remote sensor that was used to remotely control the Hong Kong robot also detected infrared sources in the Irvine demonstration site. The two relative locations of the two detected infrared sources, shown as two red dots, were selected for display on the upper right of the UDOP.

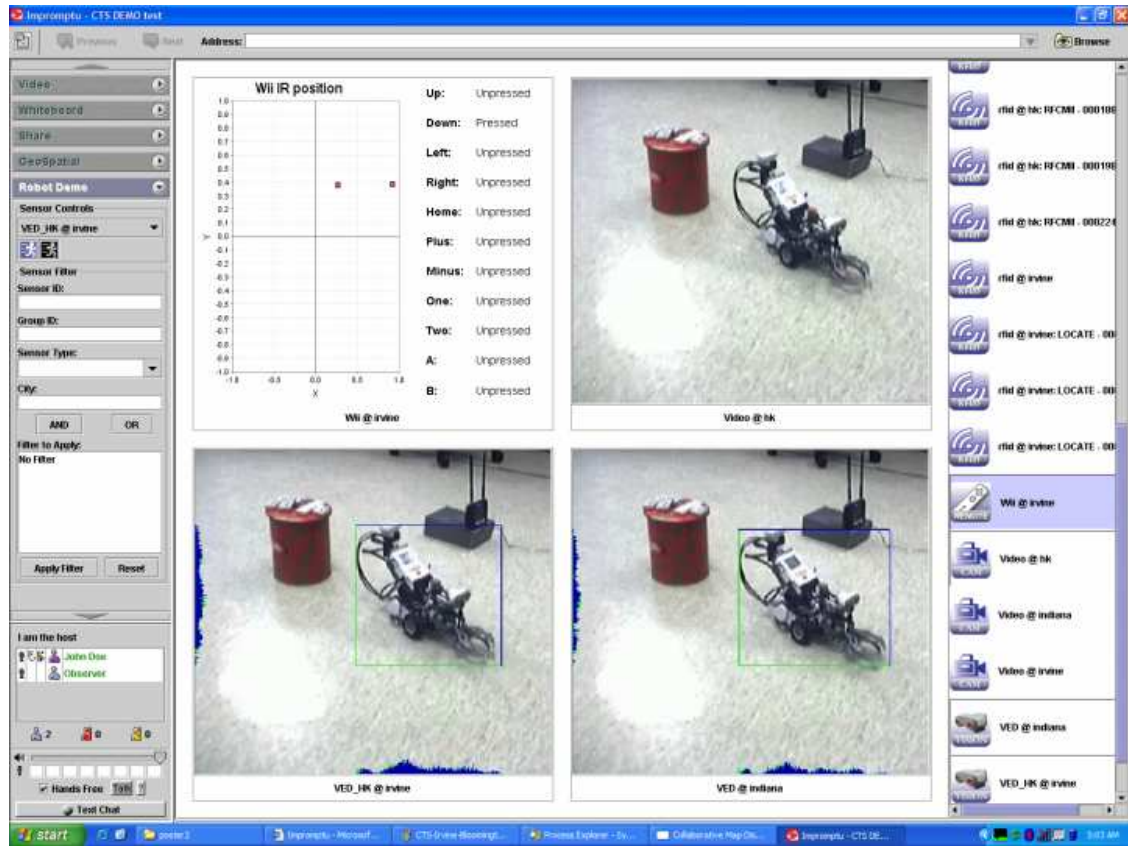


Figure 5-10 A video, Wii Remote and motion edge detection operational picture

In Figure 5-11 a UDOP about the environmental situation in the Irvine demonstration site was dynamically created. On the upper left the 7 RFID tag signal strength streams were displayed live while on the lower left the streams of data from the three active sensors – Sound, Light and Gyroscope sensors - carried by the Irvine deployed Lego Tribot robot was shown. On the upper right, and the VED sensor that was deployed in Irvine detected motion in the video stream from Irvine. The motion area was shown as a rectangular bounding box on the lower right. Even though the two robots were not in motion at the time, the minimum rectangular bounding box showed in the VED output stream on the lower right of the UDOP indicated that the two people at the far end of the demonstration site were moving.



Figure 5-11 A UDOP for situational awareness around the Irvine demonstration site

The next two figures illustrate the situational awareness capability of the SCGMMS prototype includes the awareness of disconnected resources. Figure 5-12 shows the same environment situational awareness UDOP similar to that of Figure 5-10 but on the sensor list on the right hand side, it also alerts in red a disconnected GPS sensor in Irvine. In the current implementation, SCGMMS updates sensor status every few seconds. In Figure 5-13 the sensor list illustrates that the Lego Humanoid and Tribot and GPS sensor in Irvine were disconnected from the grid.

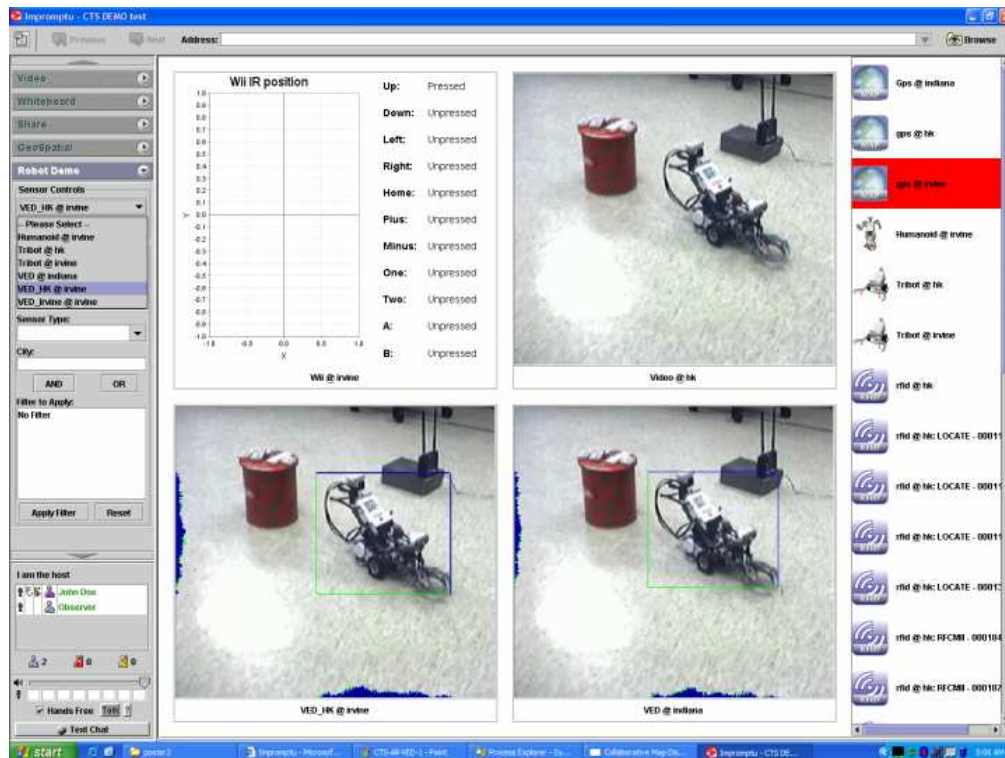


Figure 5-12 Live alert of disconnected GPS sensor

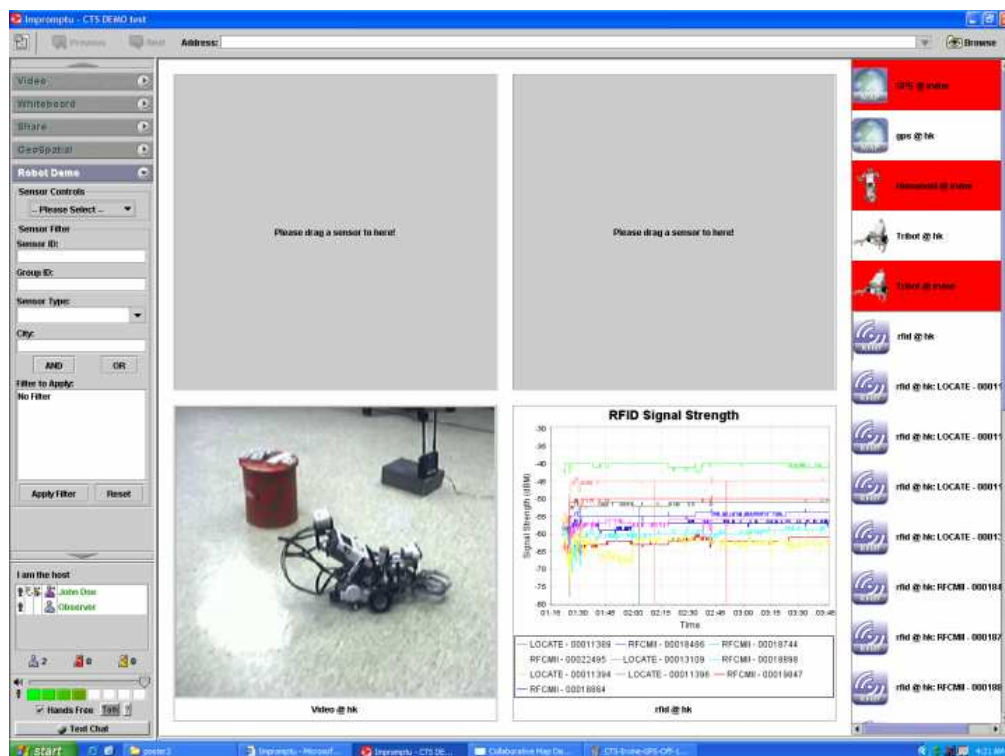


Figure 5-13 Disconnected sensors in Irvine

Operational Picture

The operational picture is composed by some operators, who are located in different geo-spatial locations (e.g. US and Hong Kong). They have to collaboratively control two NXT robots in two remote locations in order to take various environmental data of that location.

5.2 A Sample Illustration of Using SCGMMS API for UDOP Applications

After CTS 2008 SCGMMS was further improved to support the concept of hierarchical sensor groups and advanced UDOP capabilities including UDOP service management – create, delete, update with annotation and tracking of UDOPs.

A sample UDOP application with sensors deployed geographically similar to that of the demonstration in Supercomputing 2007 (Reno, San Francisco, Bloomington, Hong Kong) was created to illustrate the new capabilities. Unlike the demonstration in Supercomputing 2007 Lego robots and RFID sensors were added this time.

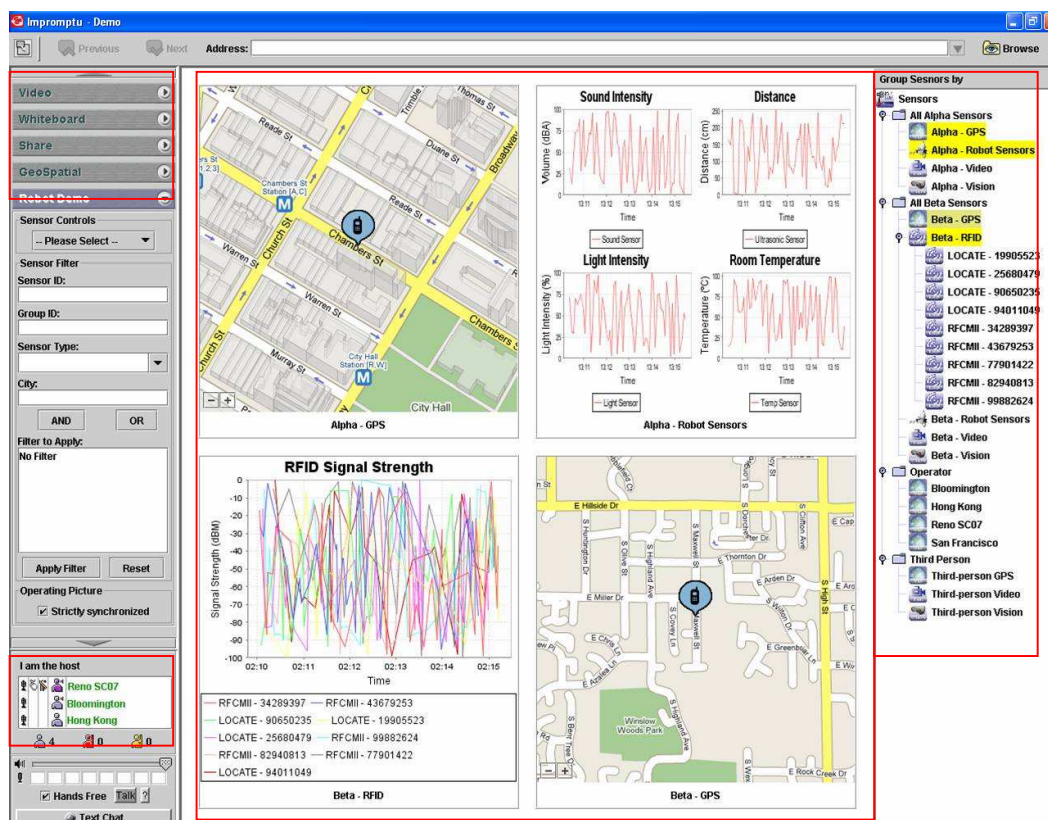


Figure 5-14 A collaborative UDOP user-interface

Figure 5-14 above shows the application which is developed based on the UDOP requirements defined. It is composed by 4 main components:

Sensor Hierarchy

On the right hand side all sensors in the operational environment is shown in hierarchies. The current implementation pre-defines the group-by attributes. Future enhancement could support user-defined hierarchies.

Each of the sensors provides a stream of raw data. Different types of sensor are displayed by different icons. Sensors which are closely related together with each other can be grouped together for easy manipulation and navigation. Notice that the application obtains the type and relationship among sensors by the reading the metadata which defines the properties of sensor.

Presentation Area

The presentation area contains four panels; each of them can display data from a sensor. In order to display data of a sensor, the user has to drag and drop the icon of the sensor from the sensor list to one of the panels.

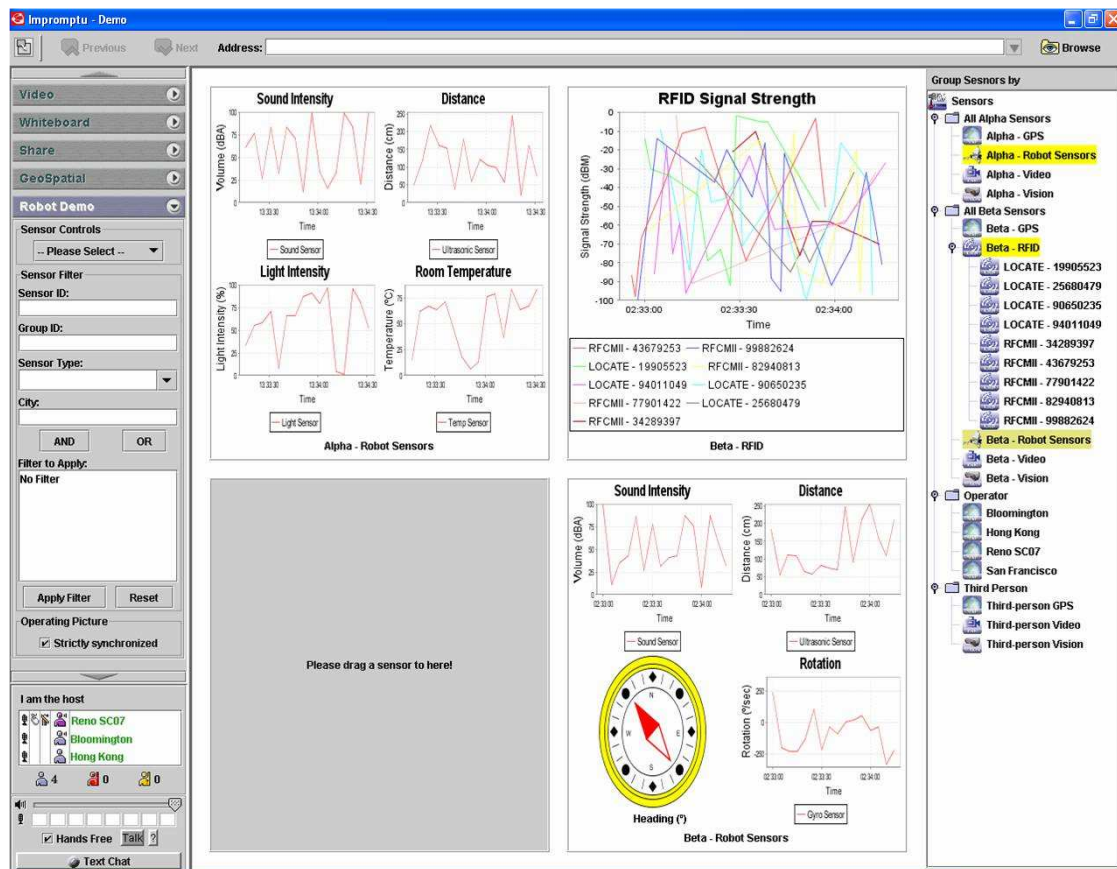


Figure 5-15 A UDOP composed by selecting from an extensible list of sensor streams

Figure 5-15 above shows data of two NXT robots and signal strength collected by an RFID reader visualized through 2D plots, progress bars and virtual compass.

Participants

The application itself supports collaboration among multiple participants. Every participant shares a common view on the operational picture using a strictly synchronous model. All participants are displayed in the lower left hand side of the screen.

Switching Operational Pictures

Given that the operational picture is already defined, it can be further broken down into smaller operational pictures which serve different purposes in the main operational picture. On the top left hand side there is a list of control buttons. Each of them supports one of the types of live operational pictures. For example, the “Video” control button is used to display the web-cameras of all the operators (Figure 5-16).

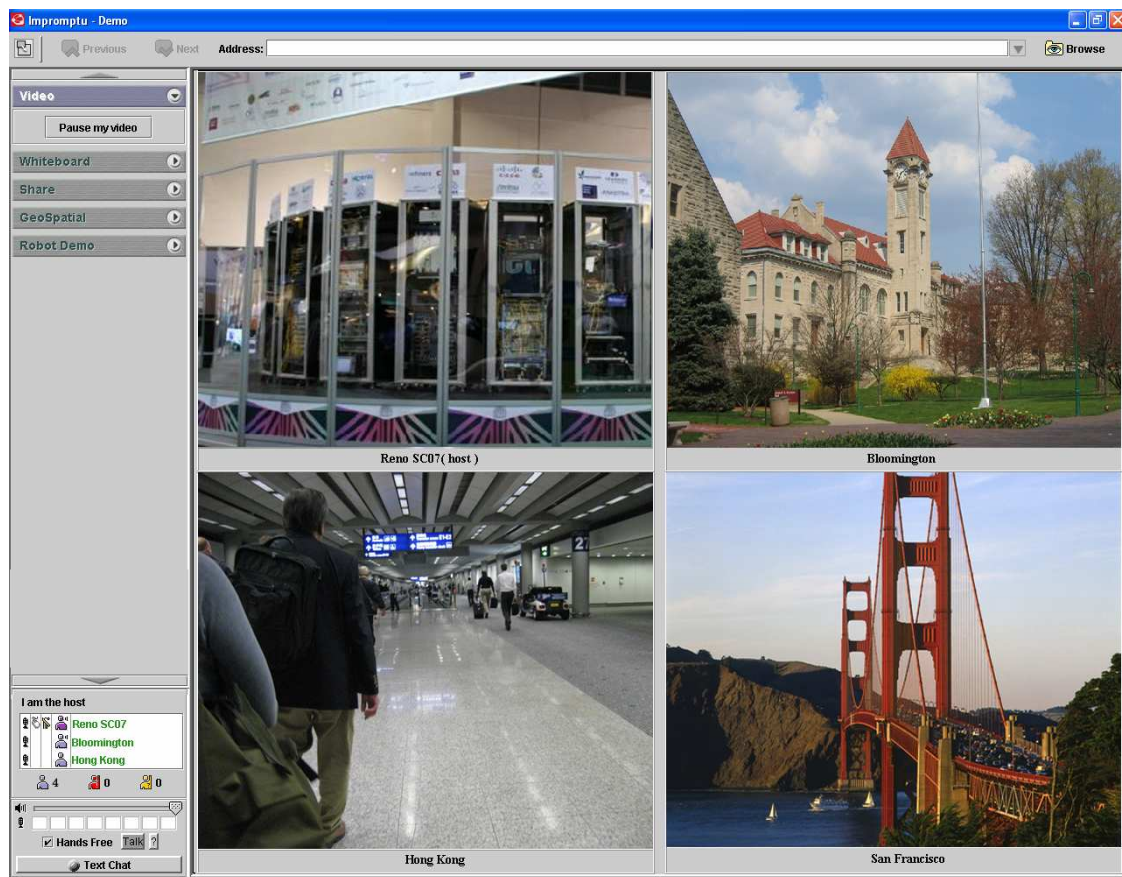


Figure 5-16 A Video Sharedlet shares four live feeds from four different cities

The “GeoSpatial” control button is used to show the geo-spatial locations of all operators and sensors. In this case, as is shown in Figure 5-17, real-time locations of GPS sensor streams are displayed and shared on a 2D world map based on Google map.

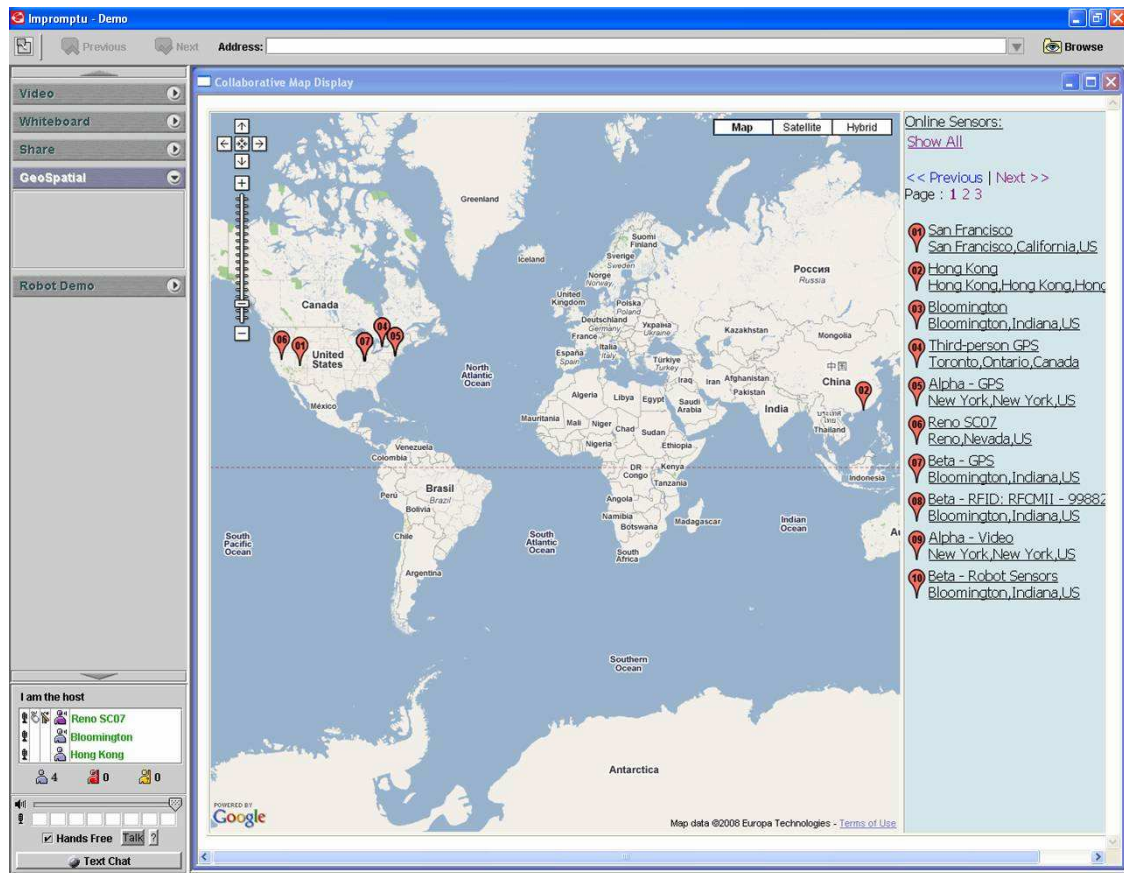


Figure 5-17 A Geo-spatial sharedlet integrating real-time GPS sensor streams

The applications can be evaluated with the following basic UDOP capabilities – distillation, transformation and aggregation.

Distillation

The “drag and drop” and filtering mechanism of the application allows the user to focus on a specific item of interest within an operational picture, while keeping items not of principal interest hidden. A user can even emphasize data from a single sensor by double-clicking one of the 4 panels and get full-screen display. Figure 5-18 is a full-screen display of RFID signal strengths after clicking on the upper right panel in Figure 5-15.

The presentation of focused sensor is optimized based on nature of the data. Without meta-data support of SCGMMS, distillation of data is much harder.

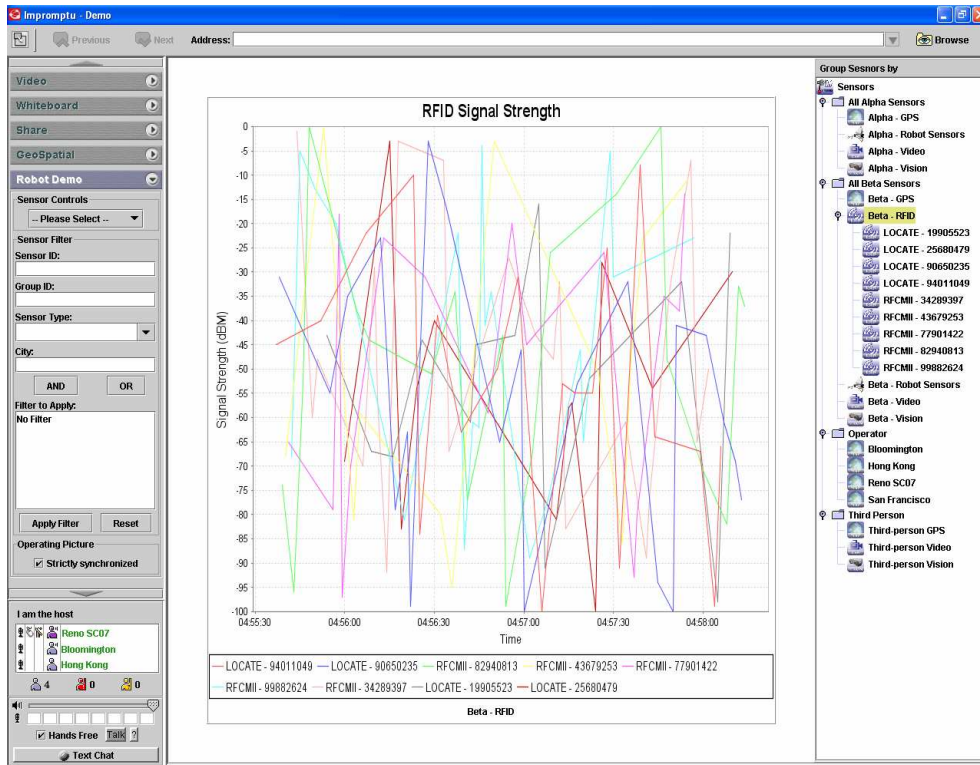


Figure 5-18 A UDOP shows and shares RFID signal strength of ten active RFID tags

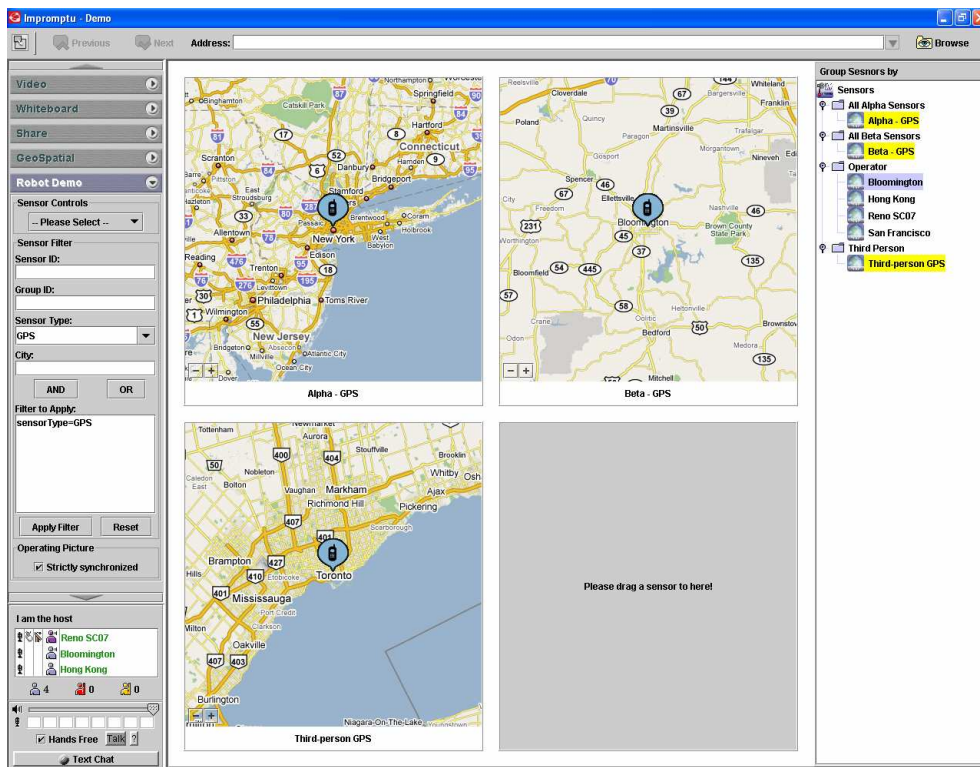


Figure 5-19 A UDOP shows the geo-spatial locations of deployed robots and an observer

Transformation

The application supports transformation of data retrieved from a particular sensor. The transformed data is more suitable for consumption by some special target audience. SCGMMS supports this kind of transformation by providing “service-oriented sensors”. This type of sensor does not read environmental data by itself. Instead, it takes data stream from another sensor based on user request, and transforms the data based on the nature of service it provides. The transformed data is returned to the target audience in a real-time stream.

The figure below shows two video streams captured by two Nokia N800 Internet Tablet PC and their corresponding processed video streams which emphasize the moving region in the captured video.

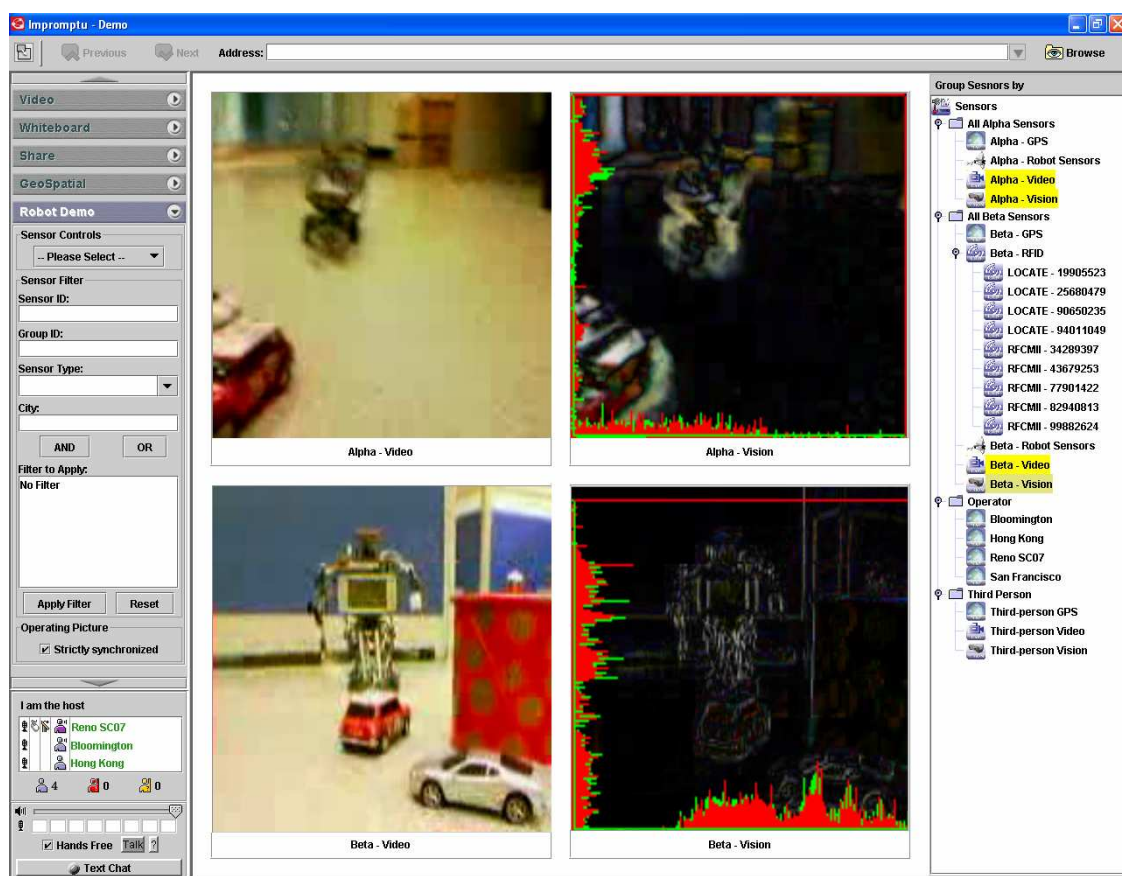


Figure 5-20 Transformation of live video into live edge-detected video

Aggregation

The application supports aggregation of data from multiple sensors into a single view. A sample application of aggregation is to display data from sensors which are attached to a specific NXT Robot.

Figure 5-21 below shows the aggregated view of a NXT Robot with all sensors attached to it. In this case, the sensors are the four NXT Robot sensors, an attached GPS sensor, a Webcam video sensor and a VED (Video Edge Detection) sensor.

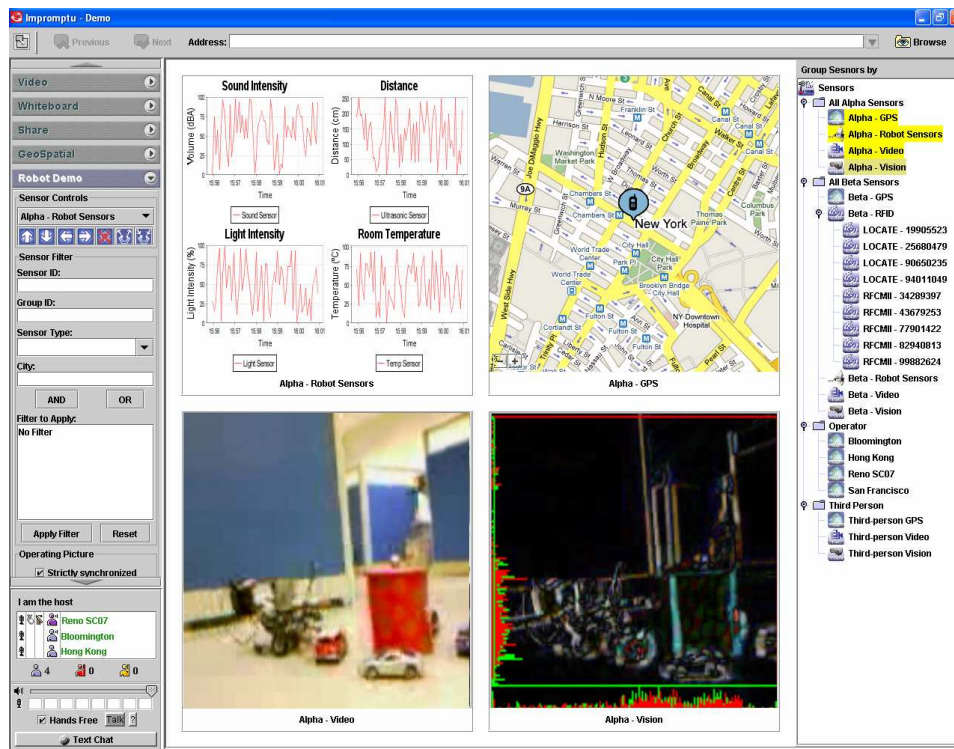


Figure 5-21 Aggregated view of an NXT Robot

Figure 5-22 to 5-25 illustrate other on-the-fly constructed UDOPs simply by drag-and-drop of sensors deployed to the presentation panels.

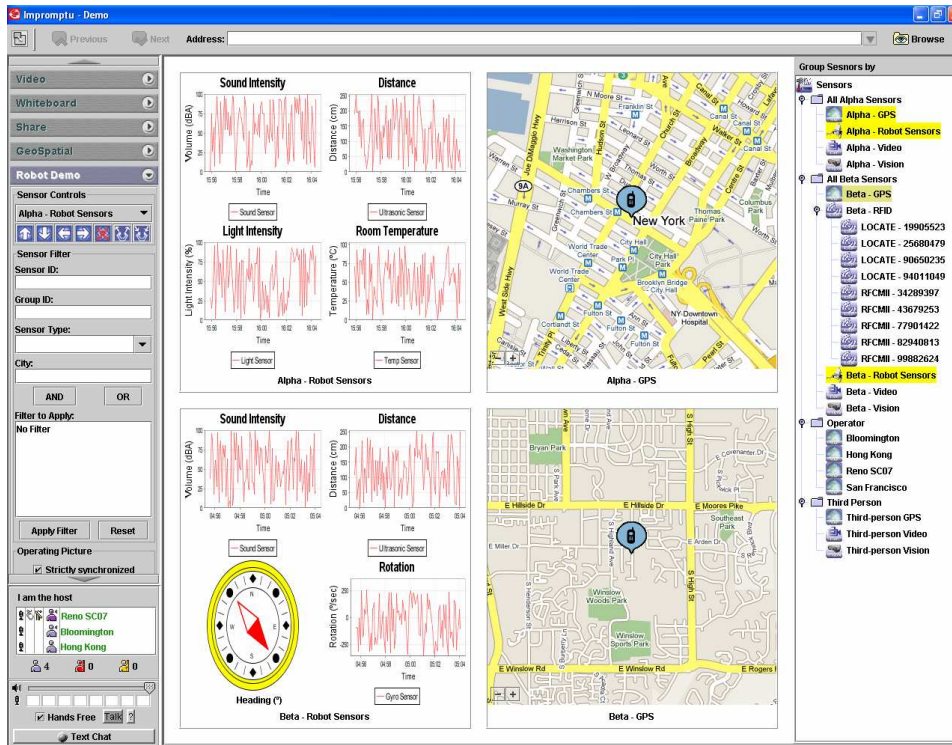


Figure 5-22 A UDOP shows and shares the eight real-time sensor streams carried by 2 robots with their respective geo-spatial information

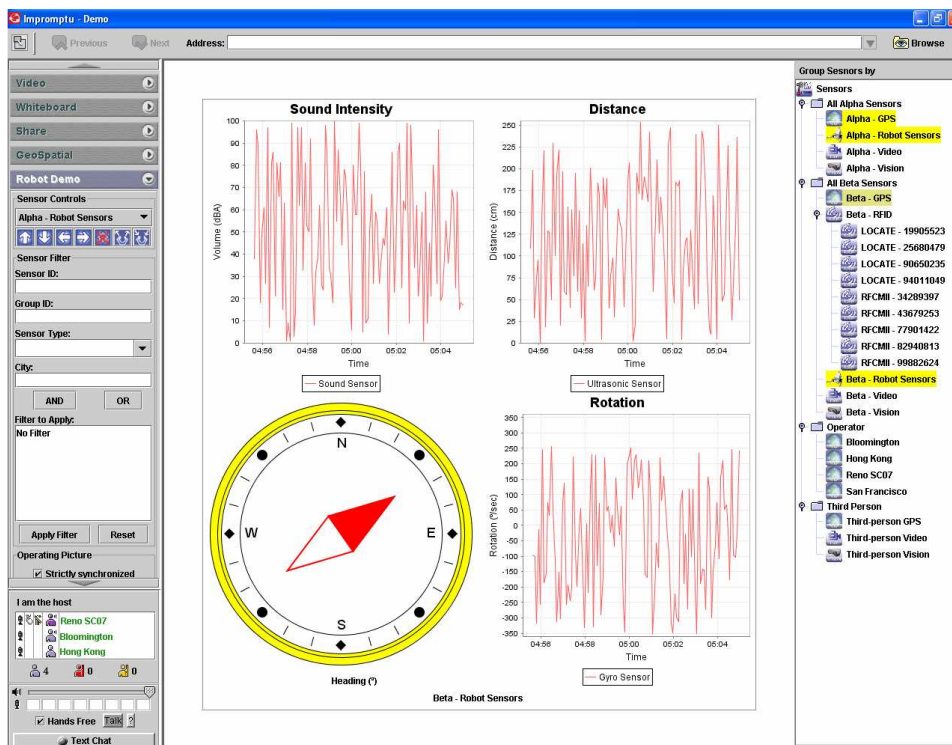


Figure 5-23 A UDOP shows and shares 4 sensor streams delivered by a Tribot robot

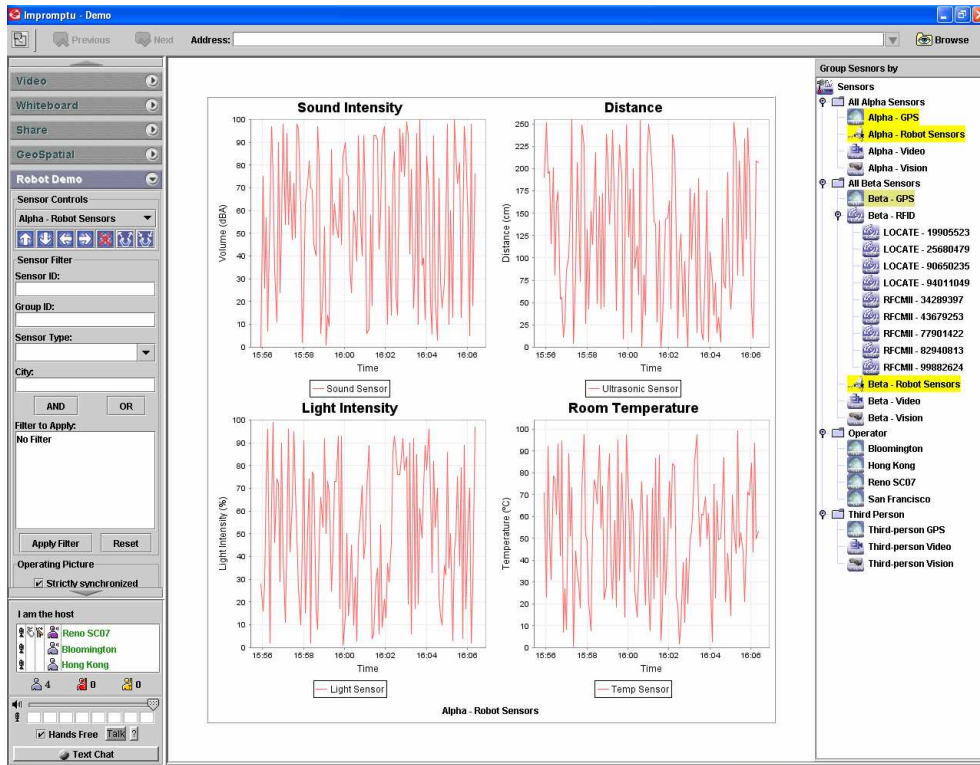


Figure 5-24 A UDOP shows and shares 4 sensors streams delivered by a Humanoid robot

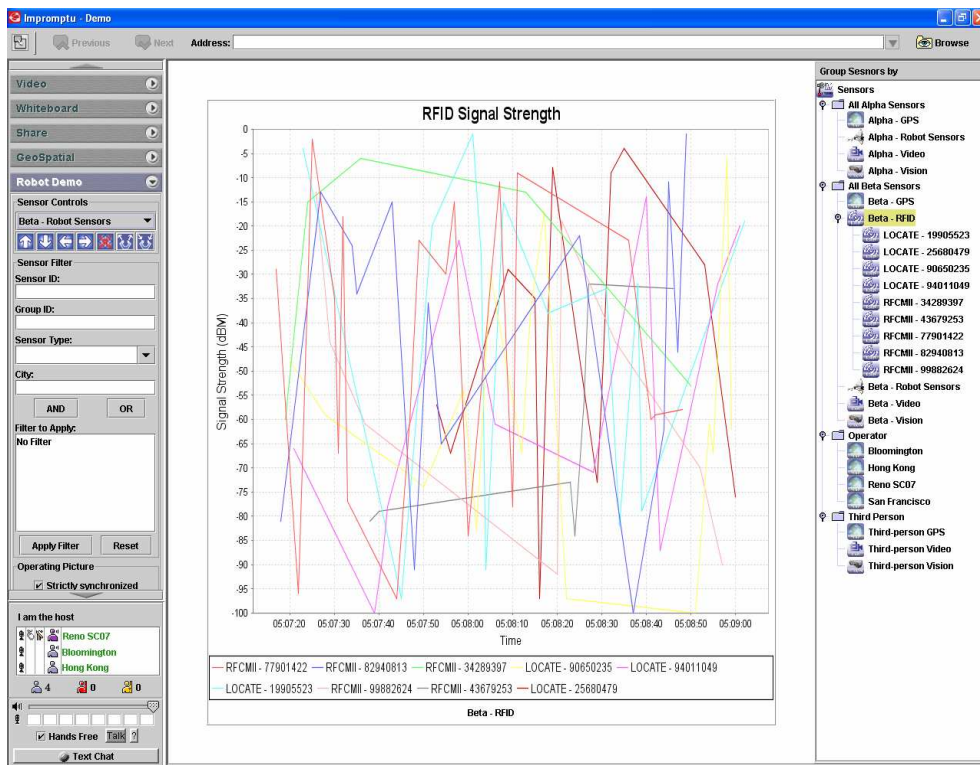


Figure 5-25 A UDOP shows and shares RFID signal strength of ten active RFID tags

6 Conclusions

We have designed and built an Enabling and Extensible Collaborative Sensor-Centric Grid Framework that supports UDOP/COP using SensaaS (Sensor as a Service). The key Software Systems and Modules are ready for use in demonstrating layered Sensor Grids and in adding new sensors and filter modules. In a grid of grids context, we integrated the following component grids:

- Sensor display and control. Here a sensor is a time-dependent stream of information with a geo-spatial location. Note that a static electronic entity is a broken sensor with a broken GPS! i.e. a sensor architecture applies to any electronic entity.
- Filters for GPS and video analysis (Compute or Simulation Grids)
- Earthquake forecasting
- Collaboration Services
- NetOps Situational Awareness Service

Features of SCGMMS include:

- An API for third-party legacy or new applications to easily acquire grid situational awareness.
- An API for sensor developers to easily integrate sensors with collaboration sensor grid to enhance situational awareness.
- A Grid Builder Management System to build, deployment, management, monitor sensor and general grids.
- Examples of integrating filter (compute) and collaboration grids with Sensor Grids in Grid of Grids scenario
- A NetOps Situational Awareness Sensor-Grid Demo Client.
- An Impromptu Sensor-Grid Demo Client with support for UDOP and Earthquake Science.

Our project reached several important conclusions:

1. Grid technology supports layered sensor networks with high performance using approximately $MN/1000$ brokers for M small (i.e. producing messages of size less than 10 kbytes each) sensors producing an average of N messages per second
2. The Sensor Grid can be integrated with a collaborative decision support environment
3. General filters can be defined as Grid services
4. The Grid framework supports a broad definition of sensor that includes any device that receives and returns information; demonstrated devices include environmental sensors, GPS, RFID, Robot and Game controllers and audio/video devices
5. The Sensor Grid can be integrated with the NetOps Network Operations tool; this integration is possible through well defined service interfaces
6. The SensaaS (Sensor as a Service) approach is successful allowing architectural integration of sensors in Grids with SaaS (Software as a Service) used for other capabilities

7. We successfully followed the significant changes in commercial distributed middleware and correctly focused on key concepts (use of services and message oriented middleware) that are still central
8. We identified importance of hierarchical topics in the publish-subscribe infrastructure and implemented in core technology but not in Sensor Grid
9. We were able to integrate portable VPN software to mitigate impact of firewalls and provide additional security; limitations in current open source VPN software prevented useful deployment in sensor Grid
10. Sensor grids of similar architecture can support DoD layered sensors as well as non military applications such as Earthquake and Environmental sensor networks
11. Security model developed and tested in point to messaging with modest overheads. However we did not tackle collective security for optimal support of layered sensors
12. It is straightforward to integrate Geographical Information Systems including Google and Microsoft clients as well as OGC (Open Geospatial Consortium) services such as WFS Web Feature Service which we extended from batch to streaming mode
13. Our current deployments do not have sufficient data traffic to stress our Grids, We have developed several performance enhancements for OGC services that could be important in future
14. Two key enhancements developed for Impromptu collaboration grid
 - o Collaborative groups supporting sub grids and communities of interest
 - o Hybrid shared display allowing dynamic choice of codec to be used when sharing applications

7 Recommendations

The Anabas Grid of Grids Net-centric framework prototype for building, deploying and managing general sub-grids has been developed with the successful delivery of an enabling collaborative sensor-centric grid middleware as a testbed to support the exploration and operational demonstration of the vision of Layered Sensing with robust collaboration and trust capabilities. The collaborative sensor-grid middleware enables easy integration with any systems or systems of systems on one end and extensibility of sensor and computational services on the other for flexible aggregation and collaboration of multi-dimensional global operational pictures and trustworthiness. This progress allows us to suggest several follow-on activities divided into broad areas – Layered Sensor Grids, Trusted Sensing, Grid of Grids and Commercialization. We give details in these areas below.

Layered Sensor Grid (i.e. collections of sensors)

- Extend current point to point security model to support collections and layers of sensors
- Investigate collective trust algorithms and services that use cross validation to enhance trust and concatenate trust, reliability and other data from sensors. One important set of services involves a database of trust metrics (as a set of time series) linked to services to analyze them (say using Hidden Markov methods) to give an estimate of current trust and projected reliability i.e. future trust.
- Design and develop sensor management services that can be used to task coordinated groups of sensors. These would use Grid workflow for coordination.
- Exploit new hierarchical topics in secure messaging subsystem
- Integrate other related systems such as NetOps and XCAT with layered sensors as a particular Grid within the Grid of Grids.
- Investigate a Web 2.0 style interface for users to define layering and additional resources of interest for their UDOP

Trusted Sensing (at level of individual sensors)

- Work with AFRL on extending capabilities of existing sensors such as RFID, GPS, Lego Robot-based, Wii, Nokia N8xx, Web-cams. Explore addition of other generic (non military) types such as cell phones, RSS/Atom feeds, Blogs and Twitter.
- Work with AFRL on supporting new sensors and new trust mechanisms with associated services and metadata. Extend the Anabas sensor framework SensaaS as needed and support implementation of sensors in testbed. This work includes extension of capabilities of current sensors. We expect most work on individual sensors and their trust will be performed by others and our responsibility will be common collective services (second item in Layered Sensor Grid topics above) and supporting the integration of this other work into Grid of Grids

- Identify new sensor related services of interest to AFRL; for example particular data fusion or analysis algorithms for sensor types of interest.

Grid of Grids

- Investigate security architecture needed to support trusted sensor grid including cloud deployment. Deploy enhanced framework.
- Investigate fault tolerance architecture needed to support trusted sensor grid including cloud deployment. Deploy enhanced framework.
- Investigate systematic use of virtual machine technology (Xen, VMware) for Grid service deployment. This complements Grids that virtualize system by virtualizing hosting nodes and allow a more powerful Grid builder model. The performance implications would be initial research
- Research Cloud Implementations of Grid Components I: Extend Grid Builder to deploy Grid Components on Clouds – including Amazon EC2 and small NSF TeraGrid cloud available to us.
- Research Cloud Implementations of Grid Components II: Measure and evaluate performance impact of cloud – especially on messaging substrate. Look at impact of federated clouds with different components of a given Grid deployed on different clouds.
- Add high performance metadata service based on WS-Context to those supported by Grid Builder.
- Quantify with AFRL guidance the “timeliness” of systems i.e. the performance of system measured in a simulated environment with characteristics similar to that expected in DoD use. The initial task here is defining and implementing the simulated environment with realistic bandwidth and latency characteristics. Measuring impact of trust mechanisms on performance would be a focus.
- Develop an Intranet Web 2.0 annotation service allowing tagging of services and electronic resources developing a rich user customizable environment. This environment can be searched to retrieve services and documents of interest to user. Tags are stored as part of system metadata. The Grid Builder will deploy Grids based on discovered services and electronic resources. Provide a secure Facebook style user profile compatible with Open Social Interface.
- Deploy a small cloud attached to AFRL Testbed. This involves investigating various IaaS (Infrastructure as a Service) and PaaS (Platform as a Service) software environments and installing on a small 4-8 node cluster.
- Develop a report surveying sensor nets, webs and grids in other communities including commercial (e.g. Microsoft Ocean network, Nokia cell phone), government (e.g. Ubiquitous City project in Korea) and academic projects (as many NASA sensor web projects, personal health monitoring).

Commercialization

We intend a three-prong strategy:

- Work with Ball and AFRL to get input for DoD application requirements for an integrable Grid situational awareness product.
- Harden SBIR result prototype to seek In-Q-Tel type of funding to commercialize and customize the prototype for Home Land Security applications.
- Commercial mobile solution applications for social networks with large number of sensors like the iPhone or Google phone.

8 References

A complete set of references was prepared for the phase I SBIR. It is available at <http://grids.ucs.indiana.edu/ptliupages/publications/gig/DODGridReferences.pdf> and is included here as reference 1 to 249, and additional references for cloud computing in reference 250 – 255.

1. [AccessGrid] DoE Access Grid Collaboration Environment <http://www.accessgrid.org/>
2. [ActiveBPEL] ActiveBPEL Open Source workflow engine <http://www.activebpel.org/>
3. [AFEI] AFEI (Association for Enterprise Integration) NetCentric Enterprise Services Workshops http://www.afei.org/news/NCES_Workshops.cfm
4. [Aktas04A] Mehmet Aktas, Galip Aydin, Andrea Donnellan, Geoffrey Fox, Robert Granat, Lisa Grant, Greg Lyzenga, Dennis McLeod, Shrideep Pallickara, Jay Parker, Marlon Pierce, John Rundle, Ahmet Sayar, and Terry Tullis *iSERVO: Implementing the International Solid Earth Research Virtual Observatory by Integrating Computational Grid and Geographical Information Web Services* Technical Report December 2004, To be published in Special Issue of Pure and Applied Geophysics (PAGEOPH) for Beijing ACES Meeting July 2004, http://grids.ucs.indiana.edu/ptliupages/publications/ISERVO_ACES_PAGEOPH.pdf.
5. [Anabas] Anabas Collaboration Environment. <http://www.anabas.com>
6. [Apache] Web site <http://www.apache.org>.
7. [Axis] Apache Axis Web Services Infrastructure <http://ws.apache.org/axis/>
8. [Aydin03A] Galip Aydin, Harun Altay, Mehmet S. Aktas, M. Necati Aysan, Geoffrey Fox, Cevat Ikibas, Jungkee Kim, Ali Kaplan, Ahmet E. Topcu, Marlon Pierce, Beytullah Yildiz, Ozgur Balsoy *Online Knowledge Center Tools for Metadata Management*, DoD HPCMP Users Group Meeting Seattle June 9-13 2003, <http://grids.ucs.indiana.edu/ptliupages/publications/OKCUGC.pdf>
9. [Balsoy02A] Ozgur Balsoy, Mehmet S. Aktas, Galip Aydin, Mehmet N. Aysan, Cevat Ikibas, Ali Kaplan, Jungkee Kim, Marlon Pierce, Ahmet Topcu, Beytullah Yildiz, and Geoffrey Fox *The Online Knowledge Center: Building a Component Based Portal*, Proceedings of the International Conference on Information and Knowledge Engineering, Las Vegas June 2002, <http://grids.ucs.indiana.edu/ptliupages/publications/OKCPaper1x1.pdf>
10. [Barrett2001] C. Barrett, R. Beckman, K. Berkbighler, K. Bisset, B. Bush, K. Campbell, S. Eubank, K. Henson, J. Hurford, D. Kubicek, M. Marathe, P. Romero, J. Smith, L. Smith, P. Speckman, P. Stretz, G. Thayer, E. Eeckhout, and M.D. Williams. TRANSIMS: Transportation Analysis Simulation System. Technical Report LA-UR-00-1725, Los Alamos National Laboratory Unclassified Report, 2001. An earlier version appears as a 7 part technical report series LA-UR-99-1658 and LA-UR-99-2574 to LA-UR-99-2580. See <http://transims.tsasa.lanl.gov/> and <http://www.transims.net/>.

11. [Barrett2002] C. Barrett, S. Eubank, M. Marathe, H. Mortveit and C. Reidys. *Science and Engineering of Large Scale Socio-Technical Simulations*, Proc. 1st International Conference on Grand Challenges in Simulations held as a part of Western Simulation Conference, San Antonio Texas, 2002, (2002).
12. [Barrett2004A] C. Barrett, S. Eubank, V. Anil Kumar, M. Marathe. *Understanding Large Scale Social and Infrastructure Networks: A Simulation Based Approach*, in SIAM news, March 2004. Appears as part of Math Awareness Month on The Mathematics of Networks.
13. [Barrett2004B] C. L. Barrett, M. Drozda, M. V. Marathe, S. S. Ravi and J. P. Smith, *A Mobility and Traffic Generation Framework for Modeling and Simulating Ad hoc Communication Networks*, Journal of Scientific Programming, Vol. 12, No. 1, 2004, pp. 1–23. (A preliminary version appeared in Proc. ACM Symposium on Applied Computing (SAC) – Special Track on Spatial Modeling, Madrid, Spain, March 2002, pp. 122–126.)
14. [BEEP] BEEP framework for building application protocols
<http://www.beeppcore.org/>
15. [Berman03A] *Grid Computing: Making the Global Infrastructure a Reality* edited by Fran Berman, Geoffrey Fox and Tony Hey, John Wiley & Sons, Chichester, England, ISBN 0-470-85319-0, March 2003. <http://www.grid2002.org>.
16. [Bernholdt98A] Bernholdt, D., Fox, G., Furmanski, W., Natarajan, B., Ozdemir, H., Ozdemir, Z., and Pulikal, T., "WebHLA - An Interactive Programming and Training Environment for High Performance Modeling and Simulation", in *Proceedings of the DoD HPC 98 Users Group Conference*, April 30, 1998.
<http://www.new-npac.org/users/fox/documents/furmpapers/paper20.html>
17. [Bernholdt98B] Bernholdt, D., Chappell, P., Fox, G., Furmanski, W., Kasthuril, D. Krishnamurthy, G., Nair, S., Ozdemir, H., Ozdemir, Z., Rangarajan, K., and Snively, K., "Parallel and Metacomputing Support for CMS-Comprehensive Minefield Simulation". Demonstration Handout, Supercomputing 98, Orlando, FL, November 7-13, 1998. <http://www.new-npac.org/users/fox/documents/furmpapers/cms-handoutpaper5.html>
18. [Birman05] Ken Birman, Robert Hillman, Stefan Pleisch, *Building network-centric military applications over service oriented architectures* SPIE Conference on DEFENSE TRANSFORMATION AND NETWORK-CENTRIC SYSTEMS at Orlando Florida 31 March 2005
http://www.cs.cornell.edu/projects/quicksilver/public_pdfs/GIGonWS_final.pdf
19. [BIRN] The Biomedical Informatics Research Network (BIRN)
<http://www.nbirn.net/>.
20. [Bishop2003] Matt Bishop, *Computer Security: Art and Science*. Addison-Wesley, 2003.
21. [Blais04A] Curt Blais, *Semantic Web Technologies for Military M&S*,
<http://www.movesinstitute.org/Openhouse2004slides/blaisSemanticWeb.ppt>
22. [Booth2004] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. "Web Service Architecture." W3C Working Group Note, 11 February 2004. Available from <http://www.w3c.org/TR/ws-arch>.
23. [BPEL4WS] BPEL4WS: F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana, *BPEL4WS, Business Process Execution Language for*

- Web Services, Version 1.0. Available from <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>.
24. [caBIG] The cancer Biomedical Informatics Grid <https://cabig.nci.nih.gov/>
 25. [C4ISRarch] Command, Control, Communications, Computer Intelligence Surveillance Reconnaissance (C4ISR) Core Architecture Data Model Version 2.0 <http://www.fas.org/irp/program/core/fnlrprt.pdf>
 26. [CEE00] Collaborative Technology Development in the Air Force Research Laboratory Requirements for an Air Force Collaborative Enterprise Environment, AFRL January 2000 specification of CEE Architecture.
 27. [Chimera] Chimera Virtual Data System from GryPhyn <http://www.griphyn.org/chimera/>
 28. [CIM] Common Information Model (CIM) from the Distributed Management Task Force DMTF <http://www.dmtf.org/standards/cim/>
 29. [Clarke02A] Jerry A. Clarke and Raju R. Namburu, "A Distributed Computing Environment for Interdisciplinary Applications", *Concurrency and Computation: Practice and Experience* Vol. 14, Grid Computing environments Special Issue 13-15, pages 1161-1174, 2002.
 30. [Centra] Centra Collaboration Environment. <http://www.centra.com>
 31. [CMCS] Collaboratory for the Multi-scale Chemical Science (CMCS) <http://cmcs.ca.sandia.gov/index.php>
 32. [CMCSpaper] *A Collaborative Informatics Infrastructure for Multi-scale Science*, James D. Myers, Thomas C. Allison, Sandra Bittner, Brett Didier, Michael Frenklach, William H. Green, Jr., Yen-Ling Ho, John Hewson, Wendy Koegler, Carina Lansing, David Leahy, Michael Lee, Renata McCoy, Michael Minkoff, Sandeep Nijsure, Gregor von Laszewski, David Montoya, Carmen Pancerella, Reinhardt Pinzon, William Pitz, Larry A. Rahn, Branko Ruscic, Karen Schuchardt, Eric Stephan, Al Wagner, Theresa Windus, Christine Yang, *Proceedings of the Challenges of Large Applications in Distributed Environments (CLADE) Workshop*, June 7, 2004, Honolulu, HI, pp. 24-33.
 33. [CoABS-A] CoABS Grid <http://coabs.globalinfotek.com> from [CoABS-B]
 34. [CoABS-B] Darpa Control of Agent-based Systems CoABS program <http://www.darpa.mil/ipto/research/coabs/>
 35. [CoaxGrid] Coalition Agents Experiment <http://www.aiai.ed.ac.uk/project/coax>
 36. [Condor] Condor Home Page <http://www.cs.wisc.edu/condor/condorg/>
 37. [Cormac2005] Andrew Cormack, "Deploying Grids." UKERNA Technical Guides. Available from <http://www.ja.net/services/publications/technical-guides/tg-grid-deployment.pdf>.
 38. [CrisisGrid] <http://www.crisisgrid.org>
 39. [Curation-A] Seminar sponsored by the Digital Preservation Coalition and the British National Space Centre, Digital Curation: digital archives, libraries, and e-science, London, 19 October 2001, <http://www.dpconline.org/graphics/events/digitalarchives.html> (Several presentations are available from this link)
 40. [DAME] DAME Distributed Aircraft Maintenance Environment project <http://www.cs.york.ac.uk/dame/>
 41. [DC] Dublin Core Metadata Initiative, <http://dublincore.org>

42. [DDMS] Department of Defense Discovery Metadata Standard (DDMS) Version 1.2
<http://diides.ncr.disa.mil/mdreg/user/DDMS.cfm>
43. [DFDL] The Data Format Description Language (DFDL) working group.
<https://forge.gridforum.org/projects/dfdl-wg/>
44. [DMSO] Defense Modeling and Simulation Office DMSO
<https://www.dmsomil/public/>
45. [DMTF] Distributed Management Task Force <http://www.dmtf.org/home>
46. [DoDescience] Geoffrey Fox, Marlon Pierce *Implications of Grids, e-Science and CyberInfrastructure for the DoD High Performance Computing Modernization Program* Technical Report September 7 2003
<http://grids.ucs.indiana.edu/ptliupages/publications/DODe-ScienceGrids.pdf>
47. [Dongarra02A] *The Sourcebook of Parallel Computing* edited by Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White, Morgan Kaufmann, November 2002
48. [EBI] EBI: European Bioinformatics Institute <http://www.ebi.ac.uk/>
49. [EDG-A] European DataGrid EDG <http://eu-datagrid.web.cern.ch/eu-datagrid/>
50. [ESS02A] Report from the NASA Earth Science Enterprise Computational Technology Requirements Workshop, held April 30 - May 1, 2002 http://esto-doc.gsfc.nasa.gov:8080/documents/Information_Systems/CT/ESE-CT-Workshop/2002/ctreqreport.pdf
51. [ExpSensorGrid] Expeditionary Sensor Grid
<http://www.nwdc.navy.mil/OperationsHome/CNAN.asp>
52. [Eubank2004] S. Eubank, H. Guclu, V.S. Anil Kumar, M. Marathe, A. Srinivasan, Z. Toroczkai and N.Wang, *Modeling Disease Outbreaks in Realistic Urban Social Networks*, Nature, 429, pp. 180-184, May (2004).
53. [Ferguson03A] Donald Ferguson, Brad Lovering, John Shewchuk, Tony Storey, *Secure, Reliable, Transacted Web Services : Architecture and Composition*, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/wsoverview.asp>
54. [FleetGrid] Fleet Battle Experiments
<http://www.nwdc.navy.mil/products/fbe/default.cfm>
55. [Foster1998] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. *Proc. A Security Architecture for Computational Grids*. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.
56. [Foster99A] Foster, I. and Kesselman, C. (eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
57. [Foster04A] *The Grid 2: Blueprint for a new Computing Infrastructure*, edited by Ian Foster and Carl Kesselman, Morgan Kaufmann 2004.
58. [Fox98A] Fox, G., Furmanski, W., Nair, S., and Ozdemir, Z., "Microsoft DirectPlay Meets DMSO RTI for Virtual Prototyping in HPC T&E Environments," in *Proceedings of the International Test and Evaluation Workshop on High Performance Computing*, June 10, 1998. Syracuse <http://www.new-npac.org/users/fox/documents/furmpapers/directplaypaper15.html>
59. [Fox98B] Fox, G., Furmanski, W., and Ozdemir, H., "Object Web (Java/CORBA) based RTI to Support Metacomputing M&S," in *Proceedings of the International Test and Evaluation Workshop on High Performance Computing*, June 10, 1998. <http://www.new-npac.org/users/fox/documents/furmpapers/owrtipaper16.html>

60. [Fox98C] Fox, G., Furmanski, W., Goveas, B., Natarajan, B., and Shanbhag, S., ``WebFlow Based Visual Authoring Tools for HLA Applications," in *Proceedings of the International Test and Evaluation Workshop on High Performance Computing*, June 10, 1998. <http://www.new-npac.org/users/fox/documents/furmpapers/authoringpaper14.html>
61. [Fox98D] Fox, G., Furmanski, W., Nair, S., Ozdemir, H., Ozdemir, Z., and Pulikal, T., ``WebHLA-An Interactive Programming and Training Environment for High Performance Modeling and Simulation," in *Proceedings of the SISO Simulation Interoperability Workshop*, S/W-98F-216, July 1, 1998. <http://www.new-npac.org/users/fox/documents/furmpapers/paper12.html>
62. [Fox98E] Fox, G., Furmanski, W. Nair, S., Ozdemir, H., Ozdemir, Z., and Pulikal, T., ``WebHLA-An Interactive Multiplayer Environment for High Performance Distributed Modeling and Simulation". October 9, 1998. <http://www.new-npac.org/users/fox/documents/furmpapers/paper7.html>
63. [Fox99A] Geoffrey C. Fox, Wojtek Furmanski, Ganesh Krishnamurthy, Hasan T. Ozdemir, Zeynep Odcikin-Ozdemir, Tom A. Pulikal, Krishnan Rangarajan and Ankur Sood, "Using WebHLA to Integrate HPC FMS Modules with Web/Commodity based Distributed Object Technologies of CORBA, Java, COM and XML", in *Proceedings of the Advanced Simulation Technologies Conference (ASTC99)*, San Diego, CA, Apr 11-15, 1999. <http://www.new-npac.org/users/fox/documents/furmpapers/paper3.html>
64. [Fox99B] G. Fox, W. Furmanski, G. Krishnamurthy, H. Ozdemir, Z. Ozdemir, T.Pulikal, K. Rangarajan and A. Sood, WebHLA as Integration Platform for FMS and other Metacomputing Application Domains, In *Proceedings of the DoD HPC Users Group Conference*, Monterey, CA, June 8-15, 1999. <http://www.new-npac.org/users/fox/documents/furmpapers/paper1.html>
65. [Fox03A] Geoffrey Fox, Dennis Gannon and Mary Thomas, *Overview of Grid Computing Environments*, Chapter 20 of [Berman03A]
66. [Fox04A] Fox, G., *WS-FlexibleRepresentation*, Community Grids Lab, Indiana University, 2004. <http://grids.ucs.indiana.edu/ptliupages/publications/presentations/jsunov04.ppt>
67. [Fox04B] Geoffrey Fox, Sang Lim, Shrideep Pallickara, Marlon Pierce "Message-Based Cellular Peer-to-Peer Grids: Foundations for Secure Federation and Autonomic Services" published in *Peer to Peer Computing and Interaction with the Grid -- a Special issue of Future Generation Computer Systems* 2004. http://grids.ucs.indiana.edu/ptliupages/publications/cellularGrid_final.pdf
68. [Fox05A] Geoffrey Fox, Shrideep Pallickara, Marlon Pierce, Harshawardhan Gadgil, *Building Messaging Substrates for Web and Grid Applications* to be published in special Issue on Scientific Applications of Grid Computing in *Philosophical Transactions of the Royal Society of London* 2005 <http://grids.ucs.indiana.edu/ptliupages/publications/RS-CGL-ColorOnlineSubmission-Dec2004.pdf>
69. [Fox05B] Geoffrey Fox *Possible Architectural Principles for OGSA-UK and other Grids* UK e-Science Core Programme Town Meeting London Monday 31st January 2005 "Defining the next Level of Services for e-Science" <http://grids.ucs.indiana.edu/ptliupages/presentations/ogsaukjan05.ppt>

70. [Fox05C] Geoffrey Fox, Alex Ho, Marlon Pierce *Grid Technology Overview and Status*, Internal Report June 2005, Anabas Inc., Community Grids Laboratory Indiana University <http://grids.ucs.indiana.edu/ptliupages/publications/gig>
71. [Fox05D] Geoffrey Fox, Alex Ho, Marlon Pierce *Grid Opportunities for the GiG and NCOW*, Internal Report July 2005, Anabas Inc., Community Grids Laboratory Indiana University <http://grids.ucs.indiana.edu/ptliupages/publications/gig>
72. [Fox05E] Geoffrey Fox, Alex Ho, Marlon Pierce *Overview of Some Grid Application Areas within DoD*, Internal Report June 2005, Anabas Inc., Community Grids Laboratory Indiana University <http://grids.ucs.indiana.edu/ptliupages/publications/gig>
73. [Fox05F] Geoffrey Fox, Alex Ho, Marlon Pierce *Implementing some Grid Application Areas within NCOW 1.1 of DoD*, Internal Report July 2005, Anabas Inc., Community Grids Laboratory Indiana University <http://grids.ucs.indiana.edu/ptliupages/publications/gig>
74. [Fox05G] Geoffrey Fox, Alex Ho, Marlon Pierce *References for DoD Grid Reports*, Internal Report July 2005, Anabas Inc., Community Grids Laboratory Indiana University <http://grids.ucs.indiana.edu/ptliupages/publications/gig>
75. [Fox05H] Geoffrey Fox, Alex Ho, Shrideep Pallickara, Marlon Pierce, Wenjun Wu *Grids for the GiG and Real Time Simulations*, Proceedings of Ninth IEEE International Symposium DS-RT 2005 on Distributed Simulation and Real Time Applications' Montreal October 10-12 2005 <http://grids.ucs.indiana.edu/ptliupages/publications/gig>
76. [Gadgil04A] Harshawardhan Gadgil, Geoffrey Fox, Shrideep Pallickara, Marlon Pierce, Robert Granat *A Scripting based Architecture for Management of Streams and Services in Real-time Grid Applications* Technical Report December 2004. Available from <http://grids.ucs.indiana.edu/ptliupages/publications/HPSearch-mgmtArch.pdf>
77. [Gannon04A] D. Gannon, J. Alameda, O. Chipara, M. Christie, V. Duple, L., Fang, M. Farrellee, G. Fox, S. Hampton, G. Kandaswamy, D. Kodeboyina, S. Krishnan, C. Moad, M. Pierce, B. Plale, A. Rossi, Y. Simmhan, A. Sarangi, A. Slominski, S. Shirasuna, T. Thomas, *Building Grid Portal Applications from a Web-Service Component Architecture* to appear in a special issue of IEEE Distributed Computing on Grid Systems 2004. <http://grids.ucs.indiana.edu/ptliupages/publications/portal-apps-arch.pdf>
78. [GapAnalysis] Geoffrey Fox, David Walker, *e-Science Gap Analysis*, June 30 2003. Report UKeS-2003-01, http://www.nesc.ac.uk/technical_papers/UKeS-2003-01/index.html. This report has 390 Grid references and an extensive glossary
79. [Gateway] Gateway Portal <http://www.gatewayportal.org/>
80. [GCE] GGF Grid Computing Environments Research Group <https://forge.gridforum.org/projects/gce-rg>
81. [GEF] The Grid Enterprise Forum <http://www.opengroup.org/gesforum/>
82. [GGF-A] Global Grid Forum <http://www.gridforum.org>
83. [GiG] DoD Global Information Grid Architectures <https://disain.disa.mil/ncow.html>
84. [GiGExecSumm] Global Information Grid Architecture Version 2: Net-Centric Operations and Warfare Executive Summary, 5 May 2003

85. [GiG-Block] Global Information Grid Architecture Version 2 white papers for the Block 2(SecDef Force Allocation Decision), 3(Homeland Defense) and 4(Southwest Asia Warfighting)
86. [Globus-A] Globus Project <http://www.globus.org>
87. [Globus-GT4] Globus Toolkit GT4 April 30 2005
<http://www.globus.org/toolkit/docs/4.0/>
88. [GlobalMMCS] GlobalMMCS Service oriented Collaboration Environment from Community Grids Laboratory <http://www.globalmmcs.org>
89. [Globus-Security] Collected Globus security papers are available from
<http://www.globus.org/alliance/publications/papers.php#Security%20Components>.
90. [GofG] Geoffrey Fox, "Grids of Grids of Simple Services" Computers in Science and Engineering July/August 2004, p84-87
<http://grids.ucs.indiana.edu/ptliupages/publications/Cisegridofgrids.pdf> or
<http://csdl.computer.org/dl/mags/cs/2004/04/c4084.pdf>
91. [GridShib] GridShib integration of Shibboleth Internet2 Security framework [Shibboleth] with the Grid <http://grid.ncsa.uiuc.edu/GridShib/> funded by NSF Middleware Initiative
92. [GridSphere] GridSphere open source portal
<http://www.gridsphere.org/gridsphere/gridsphere>
93. [Groove] Groove Desktop Collaboration Software, <http://www.groove.net/>
94. [Hayes04] Rick Hayes-Roth *Comments on NCOW RM 1.01*, October 19 2004
http://www.w2cog.org/documents/RHR_comments_re_NCOW_RM_1.01.doc
95. [Haupt03A] Tom Haupt and Marlon Pierce, *Distributed object-based grid computing environments*, Chapter 30 of [Berman03A]
96. [HLA] High Level Architecture HLA <https://www.dmsomil/public/transition/hla/> - a framework for distributed military models and simulations
97. [Horn01A] Paul Horn, IBM, 10/15/2001 presentation at the AGENDA 2001 conference in Scottsdale, AZ, *Autonomic Computing : IBM's Perspective on the State of Information Technology*,
http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf
98. [HotPage] NPACI HotPage <http://hotpage.npaci.edu/>
99. [HPCMP] High Performance Computing Modernization Program (HPCMP)
<http://www.hpcmo.hpc.mil/>
100. [HPSearch] HPSearch Web Service Scripting Interface <http://www.hpsearch.org>
101. [Humphrey2005] Marty Humphrey, Glenn Wasson, Jarek Gawor, Joe Bester, Sam Lang, Ian Foster, Stephen Pickles, Mark Mc Keown, Keith Jackson, Joshua Boverhof, Matt Rodriguez, and Sam Meder, "State and Events for Web Services: A Comparison of Five WS-Resource Framework and WS-Notification Implementations." HPDC 14, July 24-27, 2005.
<http://www.caip.rutgers.edu/hpdc2005/index.html>
102. [ICCM] Intelligence Community Metadata Working Group web site:
<http://www.xml.saic.com/icml/>
103. [IEEE1516] P1516 - Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) — Framework and Rules
<http://standards.ieee.org/board/nes/projects/1516.pdf>; there are associated IEEE standards projects.

104. [Interwise] Interwise Enterprise Communications Platform,
<http://www.interwise.com>
105. [IOTA1] Infrastructure Operations Tool Access web site:
<https://extranet.if.afrl.af.mil/iota/>
106. [IOTA2] Infrastructure Operations Tools Access (IOTA) Functional Requirements Document (FRD). Available from
https://extranet.if.afrl.af.mil/iota/10docs/IOTA_V_1_0_FRD.DOC
107. [IPG] NASA Information Power Grid <http://www.ipg.nasa.gov/>
108. [iVOA] International Virtual Observatory Alliance <http://www.ivoa.net/>
109. [JBI] Joint Battlespace Infosphere JBI.
<http://www.rl.af.mil/programs/jbi/documents/JBIVolume1.pdf>
110. [JBIGrid] Joint Battlespace Infosphere
<http://www.rl.af.mil/programs/jbi/default.cfm>
111. [Johnston03A] Bill Johnston NASA IPG, DoE Science Grid, *Implementing Production Grids*, Chapter 5 of [Berman03A]
112. [Jetspeed] Apache Portal project <http://portals.apache.org/>
113. [JMS] The Java Message Service JMS <http://java.sun.com/products/jms/>
114. [JSR168] JSR-000168 Portlet Specification for Java binding (Java Community Process) October 2003
<http://www.jcp.org/aboutJava/communityprocess/final/jsr168/>
115. [JV2020] Joint Vision 2020 <http://www.dtic.mil/jointvision>
116. [JXTA] JXTA peer to peer environment from Sun Microsystems
<http://www.jxta.org>
117. [Kepler] Kepler scientific workflow <http://kepler.ecoinformatics.org/>
118. [Kerr04] *(Service Oriented) Software Development Guidance to the Department of the Navy Project Manager* An overview by Bill Kerr, Fleet Numerical Meteorology and Oceanography Center Science and Technology Advancement Team, 5 August 2004
http://www.w2cog.org/documents/Kerr_Guidance_overview.doc
119. [KK] KnowledgeKinetics, Ball Aerospace enterprise collaboration environment in CEE architecture http://www.ball.com/aerospace/k2_home.html
120. [Krieger03] Mike Krieger Director, Information Management DASD(DCIO), OASD(NII) *NCES Net-Centric Enterprise Services* The Open Group Conference Washington DC 20-24 October 2003
<http://www.opengroup.org/public/member/proceedings/q403/krieger.pdf>
121. [Laszewski02A] Gregor von Laszewski, Mei-Hui Su, Ian Foster, Carl Kesselman, Quasi Real-Time Microtomography Experiments at Photon Sources, in [Dongarra02A]
122. [Lau04] Yun-Tung Lau , Service-Oriented Architecture and the C4ISR Framework, <http://www.stsc.hill.af.mil/crosstalk/2004/09/0409Lau.html>
123. [LCG] LCG: LHC Computing Grid, <http://lcg.web.cern.ch/LCG/>
124. [Levitt05] Bill Levitt, NCOW RM Development Group *Update on Target Technical View - Emerging Net-Centric Standards - NCOW Reference Model v1.1* The Open Group Conference January 25, 2005, San Francisco
http://www.opengroup.org/gesforum/uploads/40/6574/NCOW_TTV_V1.1_Open_Group.ppt

125. [Liberty] Liberty digital identity alliance <http://www.projectliberty.org/>
126. [Mayfield03] Terry Mayfield IDA *Net Centric Operations & Warfare Reference Model (Version 1.0)* National Defense Industry Association 3 October 2003
<http://www.afei.org/pdf/ncow/Mayfield.pdf>
127. [McQuay04] Bill McQuay, Collaborative Enterprise Technologies, presentation at AFRL-Ball Aerospace-Community Grids Laboratory meeting Dayton Aug 4 2004.
128. [Moen03A] D. M. Moen and J. M. Pullen, *Enabling real-time distributed virtual simulation over the internet using host-based overlay multicast*, in Proceedings of the IEEE/ACM Distributed Simulation-Real Time Application Symposium, 2003, pp. 30–36. http://netlab.gmu.edu/XMSF/pubs/ds-rt03_moen-pullen.pdf
129. [Morse04A] Katherine L. Morse, David L. Drake, Ryan P.Z. Brunton, *Web Enabling HLA Compliant Simulations to Support Network Centric Applications* CCRTS Command and Control Research and Technology Symposium San Diego 15-17 June 2004
http://www.dodccrp.org/events/2004/CCRTS_San_Diego/CD/papers/172.pdf
130. [MQSeries] MQSeries in IBM WebSphere <http://www-3.ibm.com/software/integration/websphere/services/>
131. [MSBinaryXML] Adam Bosworth, Don Box, Martin Gudgin, Mark Nottingham, David Orchard, Jeffrey Schlimmer, Microsoft and BEA, *XML, SOAP, and Binary Data*
http://msdn.microsoft.com/webservices/webservices/understanding/specs/default.aspx?pull=/library/en-us/dnwebsrv/html/infoset_whitepaper.asp
132. [MSSecurity] Microsoft Web Service security summary
<http://msdn.microsoft.com/webservices/webservices/understanding/specs/default.aspx?pull=/library/en-us/dnglobspec/html/wssecurspecindex.asp> and this page references both other summaries and specifications.
133. [MSTXS] Luis Felipe Cabrera, George Copeland, Jim Johnson, David Langworthy Microsoft, *Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity*
<http://msdn.microsoft.com/webservices/webservices/understanding/specs/default.aspx?pull=/library/en-us/dnwebsrv/html/wsacoord.asp>
134. [MSWSSite] Microsoft Summary of Web Service Specifications
<http://msdn.microsoft.com/webservices/webservices/understanding/specs/default.aspx>
135. [MTOM] SOAP Message Transmission Optimization Mechanism. Microsoft, IBM and BEA. <http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/>
136. [myGrid-B] Taverna myGrid Workflow
<http://mygrid.man.ac.uk/myGrid/web/components/Workflow/>
137. [myGrid-D] The myGrid Provenance Service
<http://mygrid.man.ac.uk/myGrid/web/components/ProvenanceData/>
138. [NCBI] National Center for Biotechnology Information
<http://www.ncbi.nlm.nih.gov/>
139. [NaradaBrokering] NaradaBrokering open source Messaging System
<http://www.naradabrokering.org>
140. [Netwarfare] Network-Centric Warfare <http://www.c3i.osd.mil/NCW>

141. [NCOIC] NCOIC Network Centric Operations Industry Consortium
<http://www.ncoic.org/>
142. [NCOW1.1] Global Information Grid Net-Centric Operations and Warfare Reference Model (NCOW RM) Version 1.1 (Draft) 8 November 2004
143. [NCOW1.1-B] Global Information Grid Net-Centric Operations and Warfare Reference Model (NCOW RM) Appendix B – NCOW RM Operational Description
144. [NCOW1.1-C] Global Information Grid Net-Centric Operations and Warfare Reference Model (NCOW RM) Appendix C – NCOW RM System/Services Description
145. [NCOW1.1-D] Global Information Grid Net-Centric Operations and Warfare Reference Model (NCOW RM) Appendix D – NCOW RM Target Technical View (TTV)
146. [NCOW1.1-E] Global Information Grid Net-Centric Operations and Warfare Reference Model (NCOW RM) Appendix E – NCOW RM Integrated Dictionary including Section 7 (Abbreviations and Acronyms)
147. [Netsolve] NetSolve and GridSolve network server project
<http://icl.cs.utk.edu/netsolve/>
148. [Ninf] Ninf network server project <http://ninf.apgrid.org/>
149. [NSF03A] Report of the National Science Foundation Blue-Ribbon Advisory Panel, *Revolutionizing Science and Engineering Through Cyberinfrastructure*,
<http://www.cise.nsf.gov/evnt/reports/toc.htm>
150. [OASIS] OASIS: Organization for the Advancement of Structured Information Standards <http://www.oasis-open.org/home/index.php>
151. [OGCE] Open Grid Computing Environment OGCE Portal Collaboration
<http://www.collab-ogce.org/nmi/index.jsp>
152. [OGSA] Open Grid Services Architecture (OGSA) Version 1.0
<https://forge.gridforum.org/projects/ogsa-wg/docman/>
153. [OGSAGloss] Open Grid Services Architecture (OGSA) Glossary
<https://forge.gridforum.org/projects/ogsa-wg/docman/>
154. [OGSA-DAI] OGSA-DAI Grid and Web database interface <http://www.ogsa-dai.org/>
155. [OGSA-Globus] GGF Open Grid Services Architecture
<http://www.globus.org/ogsa/>
156. [OGSIv1] OGSI Open Grid Service Infrastructure (OGSI) version 1
<http://www.gridforum.org/documents/GFD.15.pdf>
157. [Oh03A] Sangyoon Oh, Geoffrey C. Fox , Sunghoon Ko *GMSME: An Architecture for Heterogeneous Collaboration with Mobile Devices* The Fifth IEEE International Conference on Mobile and Wireless Communications Networks (MWCN 2003) Singapore in September / October, 2003.
<http://grids.ucs.indiana.edu/ptliupages/publications/mwcn2003.pdf> and
<http://grids.ucs.indiana.edu/ptliupages/projects/carousel/>
158. [Oh2005] Sangyoon Oh, Hasan Bulut, Ahmet Uyar, Wenjun Wu, Geoffrey Fox [Optimized Communication using the SOAP Infoset For Mobile Multimedia Collaboration Applications](#) Proceedings of the International Symposium on Collaborative Technologies and Systems CTS05 May 2005, St. Louis Missouri, USA.

159. [OMII] OMII UK e-Science Open Middleware Infrastructure Institute
<http://download.omii.ac.uk>
160. [openGIS] Open GIS Consortium, Inc. <http://www.opengis.org/>
161. [Pallickara03A] Shrideep Pallickara, Marlon Pierce, Geoffrey Fox, Yan Yan, Yi Huang *A Security Framework for Distributed Brokering Systems*.
http://grids.ucs.indiana.edu/ptliupages/publications/NB-SecurityFramework_acmcass.pdf.
162. [Pegasus] Globus Pegasus Planning System in Data Management: The Globus Perspective, Globus World January 2003,
http://www.globusworld.org/globusworld_web/track2/4_DataManagement1.pdf
163. [Pierce02A] Marlon. E. Pierce, Choonhan Youn, Geoffrey C. Fox *The Gateway Computational Web Portal* Concurrency and Computation: Practice and Experience in Grid Computing environments Special Issue 14, 1411-1426(2002).
<http://grids.ucs.indiana.edu/ptliupages/publications/c543finalGateway.pdf>
164. [Placeware] Placeware Collaboration Environment. <http://www.placeware.com>
165. [PST] The Practical Supercomputing Toolkit. <http://www.pstoolkit.org/index.html>.
166. [Pullen04A] J. Mark Pullen, Ryan Brunton, Don Brutzman, David Drake, Michael Hieb, Katherine L. Morse, Andreas Tolk. *Using Web Services to Integrate Heterogeneous Simulations in a Grid Environment*. To appear in Proceedings of the International Conference on Computational Science 2004, Krakow, Poland
<http://www.vmasc.odu.edu/publications/tolk/DS-GRID04-10.pdf>
167. [QuakesimCGL] The Quakesim portlet-based portal at Indiana University
<http://complexity.ucs.indiana.edu:8282>
168. [Quakesim] QuakeSim Earthquake Simulation Project Home Page
<http://quakesim.jpl.nasa.gov/>
169. [RDF-A] RDF: O. Lassila and R. R. Swick, eds. , Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999. <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>.
170. [RDF-B] RDF Schema: D. Brinkley and R.V. Guha, eds., RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft 23 January 2003.
<http://www.w3.org/TR/rdf-schema/>.
171. [RIB] Repository in a Box for web-based metadata catalog
<http://icl.cs.utk.edu/rib/>
172. [RTI] RTI or Runtime Infrastructure software implementing the HLA architecture [HLA] (interfaces) <https://www.dmsi.mil/public/transition/hla/rti/>
173. [Rycerz04] Rycerz, K.; Balis, B.; Szymacha, R.; Bubak, M.; Slood, P. *Monitoring of HLA Grid Application Federates with OCM-G* Proceedings of 8th IEEE DS-RT 2004. 21-23 Oct. 2004, Pages: 125 - 132
174. [SAM] Scientific Annotation Middleware (SAM),
<http://collaboratory.emsl.pnl.gov/docs/collab/sam/>
175. [SAML] OASIS Security Services (SAML) TC. SAML documents and specifications are available from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security .
176. [Schneier1996] Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, 1996.
177. [SemanticGrid] Semantic Grid <http://www.semanticgrid.org>

178. [SemanticWeb] Semantic Web <http://www.w3.org/2001/sw/>
179. [SensorML] Sensor Model Language (SensorML) project and specification page: <http://vast.nsstc.uah.edu/SensorML/>.
180. [SERVOGrid] Solid Earth Research Virtual Observatory <http://www.servogrid.org>
181. [SETI] SETI@Home Internet Computing <http://setiathome.ssl.berkeley.edu/>
182. [Sherman03] Sherman, A.T. and McGrew, D. A., Key Establishment in Large Dynamic Groups Using One-way Function Trees, IEEE Transactions on Software Engineering, vol. 29, NO. 5, May 2003, pp. 444-458.
183. [Shibboleth] Shibboleth Internet2 Security framework designed for university environments <http://shibboleth.internet2.edu/>
184. [Slide] Apache Slide Content Management System supporting WebDAV <http://jakarta.apache.org/slide/>
185. [SmartFrog] Hewlett Packard SmartFrog Configuration Framework <http://www-uk.hpl.hp.com/smartfrog/>
186. [SOAP] SOAP: Simple Object Access Protocol <http://www.w3.org/TR/SOAP/>
187. [SOAPInfoSet1] SOAP InfoSet described in SOAP 1.2 Primer <http://www.w3c.org/TR/2003/REC-soap12-part0-20030624/>
188. [SOAPInfoSet2] M. Gudgin, et al, "SOAP Version 1.2 Part 1: Messaging Framework," June 2003. <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>
189. [SVG] Scalable Vector Graphics SVG from W3C <http://www.w3.org/Graphics/SVG/>
190. [TeraGrid] NSF TeraGrid Project <http://www.teragrid.org>
191. [Thomas03A] Mary Thomas, Marlon Pierce, Tomasz Haupt, *Building Interoperable Portals with Web Services* Technical Report of ET011 Project October 2003 <http://grids.ucs.indiana.edu/ptliupages/publications/ET-03-011%20FY%2003%20project%20final%20rpt.doc>
192. [Taiani2005] Francois Taiani, Matti A. Hiltunen, and Richard D. Schlichting, "The Impact of Web Service Integration on Grid Performance." HPDC 14, July 24-27, 2005. <http://www.caip.rutgers.edu/hpdc2005/index.html>
193. [Tolk04A] Andreas Tolk, *XML Mediation Services utilizing Model Based Data Management*, 2004 Winter Simulation Conference, SCS, Arlington, VA, December 2004 <http://www.vmasc.odu.edu/publications/Tolk/tolka42424i.pdf>
194. [Triana-A] Triana Project <http://www.triana.co.uk/>
195. [UDDI] UDDI: Universal Description, Discovery and Integration technology from OASIS <http://www.uddi.org/>
196. [UKeS-A] UK e-Science Program <http://www.escience-grid.org.uk/>
197. [Unicore-A] UNICORE UNiform Interface to COmputing Resources <http://www.unicore.de/>
198. [uPortal] Portal from a consortium of universities <http://www.uportal.org/>
199. [Venugopal05A] Srikumar Venugopal, Rajkumar Buyya, and Kotagiri Ramamohanarao, *A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing*, Technical Report, GRIDS-TR-2005-3, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, April 21, 2005. <http://www.gridbus.org/reports/DataGridTaxonomy.pdf>

200. [VNC] Virtual Network Computing System (VNC).
<http://www.uk.research.att.com/vnc>
201. [Vogels03A] W. Vogels, *Web Services Are Not Distrubted Objects*. IEEE Internet Computing, vol. 7 (6), pp59-66, 2003.
202. [VPG] Virtual Proving Ground (VPG) <http://vpg.dtc.army.mil>
203. [VRVS] VRVS Collaboration Environment from Caltech <http://www.vrvs.org/>
204. [W2COG] W2COG World Wide Consortium for the Grid <http://www.w2cog.org/>
205. [W3CBinaryXML] W3C XML Binary Characterization Working Group
<http://www.w3.org/XML/Binary/>
206. [WebDAV] WebDAV: *Web-based Distributed Authoring and Versioning*,
<http://www.webdav.org/>
207. [WebDAV-IETF] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen,
"HTTP Extensions for Distributed Authoring—WEBDAV." Internet Engineering
Task Force (IETF) Request for Comments 2518. Available from
<http://www.ietf.org/rfc/rfc2518.txt>.
208. [WebEx] WebEx Collaboration Environment. <http://www.webex.com>
209. [Weerawarana05A] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann,
Tony Storey, Donald F. Ferguson, *Web Services Platform Architecture: SOAP,
WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*,
Prentice Hall March 22, 2005, ISBN: 0-13-148874-0
210. [Wong98] Wong, C. K. and Gouda, M. and Lam, S. S., Secure Group
Communication Using Key Graphs, ACM SIGCOMM, 1998.
211. [workflow] Grid workflow is summarized in GGF10 Berlin meeting
<http://www.extreme.indiana.edu/groc/ggf10-ww/> with longer papers to appear in a
special issue of Concurrency&Computation: Practice&Experience at
<http://www.cc-pe.net/iuhome/workflow2004index.html>. Editorial is Dennis Gannon
and Geoffrey Fox *Workflow in Grid Systems*
<http://grids.ucs.indiana.edu/ptliupages/publications/Workflow-overview.pdf>.
212. [WSA] WS-Addressing Web Services Addressing
<http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/>
213. [WS-Context] WS-Context from OASIS [http://www.oasis-](http://www.oasis-open.org/committees/download.php/9904/WS-Context.zip)
[open.org/committees/download.php/9904/WS-Context.zip](http://www.oasis-open.org/committees/download.php/9904/WS-Context.zip) November 2004
214. [WS-DM] WS-DM Web Services Distributed Management Framework (OASIS)
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm
215. [WSE] WS-Eventing.
[http://msdn.microsoft.com/library/default.asp?url=/library/en-](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/WS-Eventing.asp)
[us/dnglobspec/html/WS-Eventing.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/WS-Eventing.asp)
216. [WSFL] WSFL: Web Services Flow Language [http://www-](http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf)
[3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf](http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf)
217. [WSGrids] M. Atkinson et al., 'Web Service Grids: An evolutionary approach',
Concurrency and Computation: Practice and Experience 17, 377-389, 2005;
http://www.nesc.ac.uk/technical_papers/UKeS-2004-05.pdf (defines WS-I+)
218. [WS-I] WS-I, "Web Services Interoperability (WS-I) Interoperability Profile
1.0a." <http://www.ws-i.org>.
219. [WS-Man] WS-Management
http://www.intel.com/technology/manage/downloads/ws_management.pdf

220. [WSN] WS-Notification. <http://www-106.ibm.com/developerworks/library/specification/ws-notification>
221. [WS-Reliability] Web Services Reliable Messaging TC WS-Reliability. <http://www.oasis-open.org/committees/download.php/5155/WS-Reliability-2004-01-26.pdf>
222. [WSRF] WSRF Web Service Resource Framework http://www.oasis-open.org/committees/tc_home.php?wgabbrev=wsrf
223. [WS-RM] WS-RM Web Services Reliable Messaging Protocol (WS-ReliableMessaging) <http://www-106.ibm.com/developerworks/webservices/library/ws-rm/>
224. [WSRP] WSRP OASIS Web Services for Remote Portlets (WSRP) <http://www.oasis-open.org/committees/>
225. [WS-Security] Web Services Security: SOAP Message Security (OASIS) Standard March 2004 <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf> with WS-I Basic Security Profile May 12 2004 <http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2004-05-12.html>
226. [WS-SC] WS-SecureConversation Web Services Secure Conversation Language May 2004 <http://www-106.ibm.com/developerworks/library/specification/ws-secon/>
227. [WS-SP] Web Service Security Policy Language (WS-Policy) July 2005. Available from <http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-securitypolicy.pdf>.
228. [WS-T] Web Services Trust Language (WS-Trust) February 2005. Available from <http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-trust.pdf>.
229. [Wu04a] W. Wu, G. C. Fox, H. Bulut, A. Uyar, H. Altay, *Design and Implementation of A Collaboration Web-services system*, Journal of Neural, Parallel & Scientific Computations, Volume 12, 2004. http://grids.ucs.indiana.edu/ptliupages/publications/npsec_xgsp-final.pdf
230. [Wulf89] Wulf, William. 1989."The National Collaboratory - A White Paper in Towards a National Collaboratory". Unpublished report of a NSF workshop, Rockefeller University, NY. March 17-18.
231. [Wytzisk03] A. Wytzisk, I. Simonis, and U. Raape, *Integration of HLA simulation models into a standardized web service world*, in Proceedings of the 2003 European Simulation Interoperability Workshop, no. 03E-SIW-019, 2003. <http://ifgi.uni-muenster.de/~simonis/download/eurosiw2003.pdf>
232. [X3D] XML-enabled 3D file format superseding VRML <http://www.web3d.org/x3d/>
233. [XGSP] XML General Session Protocol developed by the GlobalMMS project [GlobalMMCS]
234. [Xie04A] Y. Xie, Y.M. Teo, W.T. Cai and S.J. Turner, *A Distributed Simulation Backbone for Executing HLA-based Simulation over the Internet*, Workshop on Grid Computing and Applications, Proceedings of the 2nd International Conference on Scientific and Engineering Computation, pp. 96-103, Singapore, June 2004. <http://www.comp.nus.edu.sg/~xieyong/publication/ICSEC2004.pdf>
235. [Xie05A] Y. Xie, Y.M. Teo, W.T. Cai and S.J. Turner, *Extending HLA's Interoperability and Reusability to the Grid*, 19th ACM/IEEE/SCS Workshop on

- Principles of Advanced and Distributed Simulation (PADS 2005) Monterey, CA, USA, June 2005. http://www.comp.nus.edu.sg/~xieyong/publication/PADS2005_xie.pdf
236. [Xie05B] Y. Xie, Y.M. Teo, W.T. Cai and S.J. Turner, *Towards Grid-Wide Modeling and Simulation*, 5th Singapore-Massachusetts Institute of Technology Alliance SMA) Annual Symposium, January, 2005. http://www.comp.nus.edu.sg/~xieyong/publication/SMA_Symposium2005_xie.pdf
237. [Xie05C] Yong Xie, Yong Meng Teo, Wentong Cai and Stephen J Turner. "Service Provisioning for HLA-based Distributed Simulation on the Grid", in *Proc of the 19th IEEE/ACM/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2005)*, pp.282-291, Monterey, California, USA, June 2005 http://www.comp.nus.edu.sg/~xieyong/publication/PADS2005_xie.pdf
238. [XMLC14N] Exclusive XML Canonicalization Version 1.0. W3C Recommendation 18 July 2002. Available from <http://www.w3.org/TR/xmlenc-core/>.
239. [XMLDSIG] XML-Signature Syntax and Processing, W3C Recommendation 12 February 2002. Available from <http://www.w3.org/TR/xmlsig-core/>.
240. [XMLENC] XML Encryption Syntax and Processing, W3C Recommendation 10 December 2002. Available from <http://www.w3.org/TR/xmlenc-core/>.
241. [XMSF1] Extensible Modeling and Simulation Framework (XMSF) Project <http://www.movesinstitute.org/xmsf/xmsf.html>
242. [XMSF2] Brutzman, Don , M. Zyda, J. Pullen, K. Morse, Extensible Modeling and Simulation Framework (XMSF), Challenges for Web-Based Modeling and Simulation, Technical Challenges Workshop, Strategic Opportunities Symposium, 22 October, <http://www.movesinstitute.org/xmsf/XmsfWorkshopSymposiumReportOctober2002.pdf>
243. [XMSF3] The MOVES Institute Open House with several definitive papers on XMSF and its Application to modeling and simulation, <http://www.movesinstitute.org/OpenHouse2004.html>.
244. [XOM] Extensible Modeling and Simulation Framework Overlay Multicast XOM <http://netlab.gmu.edu/XOM/>
245. [XOP] XML-binary Optimized Packing. Microsoft, IBM and BEA. <http://www.w3.org/TR/2005/REC-xop10-20050125/>
246. [XSBC] XSBC XML Binary Serialization Project <http://cvs.sourceforge.net/viewcvs.py/xmsf/xsbc/docs/xsbc.html> and <http://www.movesinstitute.org/Openhouse2004slides/Norbratenopenhouse2004.ppt>
247. [Yu05A] Jia Yu and Rajkumar Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, Technical Report, GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, March 10, 2005. <http://www.gridbus.org/reports/GridWorkflowTaxonomy.pdf>
248. [Zajac04] K. Zajac, M. Bubak, M. Malawski, and P. M. A. Sloot, *Towards a grid management system for HLA-based interactive simulations*, in *Proceedings of Seventh IEEE International Symposium on Distributed Simulation and Real Time*

- Applications (DS-RT 2003), S. Turner and S. Taylor, Eds. Delft, The Netherlands: IEEE Computer Society, October 2003, pp. 4–11.
249. [Zong04] W. Zong, Y. Wang, W. Cai, and S. J. Turner, *Grid services and service discovery for HLA-based distributed simulation*, in Proceedings of the IEEE/ACM Distributed Simulation-Real Time Application Symposium, 2004.
http://www.cs.unibo.it/DS-RT2004/DSRT_TPweb.htm
 250. [CloudByrantCMU] R. Byrant, *Data Intensive Cloud Computing*, Carnegie Mellon University, HPDC Cloud Panel, ACM International Symposium on High Performance Distributed Computing, 2008.
 251. [CloudQuanIBM] D. Quan, *Cloud Computing*, HPDC Cloud Panel, ACM International Symposium on High Performance Distributed Computing, 2008.
 252. [CloudReedMicrosoft] D. Reed, *Seattle: We Know About Clouds!*, HPDC Cloud Panel, ACM International Symposium on High Performance Distributed Computing, 2008.
 253. [CloudFoxIU] S. Jha, A. Merzky, G.C. Fox, *Clouds Provide Grids with Higher-Levels of Abstraction and Explicit Support for Usage Modes*, Open Grid Forum 23, Cloud Workshop, June 2008.
 254. [CloudRichardson] A. Richardson, *Being in the Clouds, Elastic Server*, Open Grid Forum 23, Cloud Workshop, June 2008.
 255. [CloudWBergerIBM] I. Wladawsky-Berger, *Keynote - Cloud Computing, Grids, and the coming Cambrian Explosion*, Open Grid Forum 22, February 2008.
 256. [RFIDWomble] P. Womble, A. Barzioov, J. Paschal, L. Hopper, A. Music, T. Morgan, R. Moore, D. Pinson, F. Schultz, M. Maston and R. Kowalik, *A tracking technology for security personnel and first responders*, in Proceedings of SPIE – Volume 5778: Sensors, Command, Control, Communications and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense IV, pp.51-56, May 2005.
 257. [RFIDNi] L. Ni, Y. Liu, Y. Lau and A. Patil, *LANDMARC: Indoor Location sensing using active RFID*, in Proceedings of IEEE International Conference in Pervasive Computing and Communications (PerCom), pp.407-415, March 2003.
 258. [RFIDYin] J. Yin, Q. Yang and L. Ni, *Learning Adaptive Temporal Radio Maps for Signal Strength-based Location Estimation*, in IEEE Transactions on Mobile Computing, 2007.
 259. [RFIDYin2] J. Yin, Q. Yang and L. Ni, *Adaptive Temporal Radio Maps for Indoor Location Estimation*, in Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom), pp.85-94, March 2005.

9 Appendice

Appendix A - User Guide for Sensor Developers

A.1 Overall SSAL Architecture

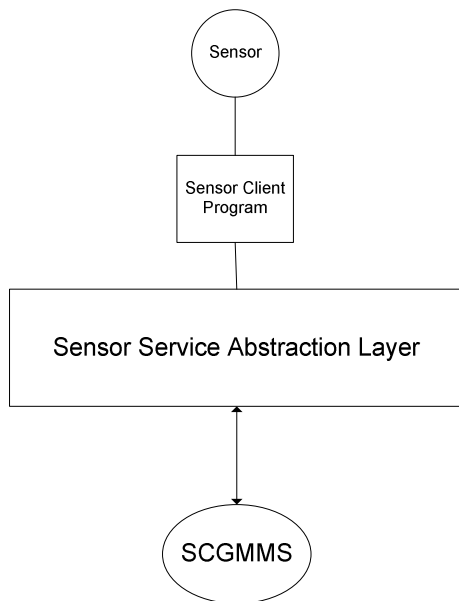


Figure 9-1 Sensor Service Abstraction Layer

One of the major objectives of SCGMMS is to allow sensor developers to create and connect their own sensors to SCGMMS so that it can be utilized by all applications based on their UDOP requirements.

Sensor Service Abstraction Layer (SSAL) provides a common interface for all kinds of sensors. Sensor developers add new sensors to SCGMMS by writing **Sensor Client Programs (SCP)** which connects to SCGMMS through libraries in SSAL.

So basically sensor developers write SCPs which is the software bridge for connecting SCGMMS with the actual sensors. A sensor developer should have programming experience using J2SE in order to use SSAL effectively.

A.2 System Requirements

SSAL is written in Java. We recommend using Java SE 5 JDK for SCP development using the SSAL.

Here is the recommended System Requirement for PC which runs application developed through the Application API

- Pentium IV 3.0 GHz Processor or above
- 512 MB RAM

- Sun Microsystems Java Runtime Environment 1.5.0 or above

A.3 Libraries

SSAL consists of a set of jar files. They include all class files necessary for communication with SCGMMS. To use it as external library, make sure that the file is in the CLASSPATH during compilation.

A.4 Detailed Descriptions

This section provides detailed descriptions on how to use different classes in SSAL to construct a SCP for connecting the sensor and SCGMMS.

A.4.1 Sensor Definition

To make the sensor understandable by SCGMMS, the first thing is to define properties of the sensor in a SensorProperty object.

Table 9-1 A sensor property object

SensorProperty	
Field:	Description
String sensorId	A string which helps identifying the sensor. Different from the unique system-generated id
String groupId	A string which identifies the name of logical group which the sensor belongs to
String sensorType	A string which represents the type of sensor (e.g. "GPS"). A list of predefined types are defined as static variables in class PredefinedType (e.g. PredefinedType.VIDEO)
int 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. A set of predefined ids can be found in class PredefinedType (e.g. GPS_ID)
String location	A string which represents the geo-spatial location of the sensor (e.g. United States, California). A list of predefined locations can be found in class PredefinedLocation. All predefined locations are associated with a latitude-longitude coordinate in the world map
boolean historical	Whether sensor data has time inter-dependence with one another
int[] sensorControl	An integer array which identifies all control messages understood by a sensor. Each control message is represented by an integer
String[] controlDescription[]	An array which contains textual description of what each control message represents. Shall align with sensorControl array
String userDefinedPropXml	For any other properties of the sensor, they can be put here as XML string. SSAL also allows sensor developers to create a class which extends UserDefinedProperty. This class can be serialized into xml string using SensorClassificationUtil.userDefinedPropertyToXml()

All these fields have to be defined as parameters to the constructor of SensorProperty. Now we are going to see how to define the SensorProperty of a NXT Humanoid Lego Robot, which has the following properties:

1. The robot comes with a software interface which connects the robot with a PC through Bluetooth. User has to provide the Bluetooth address of the robot in order to connect it with the interface
2. There are several types of NXT robots each has different physical form and perform different actions according to received control messages. Humanoid is two-legged model with 2 arms. It takes control messages to move forward and backward with legs and swing its arms continuously
3. Each robot has 4 input ports which allow 4 different types of sensors to be connected, including light, touch, sound, ultrasonic, compass, temperature and gyro sensors.

Defining Sensor Type

Firstly let define the String and integer which uniquely identifies the sensor type of NXT robot.

```
String sensorType = "NXT ROBOT";  
int sensorTypeId = 0;
```

Defining User-defined Properties

As there are some properties of the NXT robot that are quite specific, they have to be defined using the userDefinedPropXml String. In this subsection we are going to demonstrate how to write a class which represents user-defined properties by extending class UserDefinedProperty.

First of all, define a class which holds all constants needed for better code management.

```
import java.util.HashMap;  
  
public class RobotConstants {  
    // To add a support of sensor, please add it to the end  
    // Its description must be added to PREDEFINED_ROBOT_SENSORS as well  
    public static final int SENSOR_INIT_IDX = 0;  
    public static final int NONE = SENSOR_INIT_IDX;  
    public static final int LIGHT = 1;  
    public static final int TOUCH = 2;  
    public static final int SOUND = 3;  
    public static final int ULTRASONIC = 4;  
    public static final int COMPASS = 5;  
    public static final int TEMPERATURE = 6;  
    public static final int GYRO = 7;  
  
    // To add a support of robot type, please add it to the end  
    // and change TYPE_END_IDX  
    // Its description must be added to PREDEFINED_ROBOTS as well  
    public static final int TYPE_INIT_IDX = 100;  
    public static final int HUMANOID = TYPE_INIT_IDX;  
    public static final int TRIBOT = 101;
```

```

    public static final String[] PREDEFINED_ROBOT_SENSORS = new
String[]{"None", "Light", "Touch", "Sound", "Ultrasonic", "Compass",
"Temperature", "Gyro"};
    public static final HashMap ROBOT_SENSOR_2_ID = new HashMap();
    static {
        for( int i = 0; i < PREDEFINED_ROBOT_SENSORS.length; i++ ) {
            ROBOT_SENSOR_2_ID.put(PREDEFINED_ROBOT_SENSORS[i],
SENSOR_INIT_IDX + i);
        }
    }

    public static final String[] PREDEFINED_ROBOTS = new
String[]{"Humanoid", "Tribot"};
    public static final HashMap ROBOT_TYPE_2_ID = new HashMap();
    static {
        for( int i = 0; i < PREDEFINED_ROBOTS.length; i++ ) {
            ROBOT_TYPE_2_ID.put(PREDEFINED_ROBOTS[i], TYPE_INIT_IDX + i);
        }
    }
}

```

The next step is to define a class which extends `UserDefinedProperty` which holds specific details of the robot defined by constants in `RobotConstants`.

```

import
com.anabas.sensorgrid.classification.userdefinedprop.UserDefinedPropert
y;

public class RobotUserDefinedProperty extends UserDefinedProperty {

    // Bluetooth address for connecting to physical robot
    private String btAddress;

    // the type of NXT robot, e.g. Humanoid, Tribot
    private int robotType;

    // type of sensors connected to the ports
    private int[] ports;

    public RobotUserDefinedProperty(String btAddress, int robotType, int
port1, int port2, int port3, int port4) {
        this.btAddress = btAddress;
        this.robotType = robotType;
        this.ports = new int[4];
        this.ports[0] = port1;
        this.ports[1] = port2;
        this.ports[2] = port3;
        this.ports[3] = port4;
    }

    public String getBtAddress() {
        return btAddress;
    }

    public int getRobotType() {

```

```

        return robotType;
    }

    public int[] getPorts() {
        return ports;
    }
}

```

Defining Sensor Control

NXT Humanoid robot takes 5 different controls messages for leg and arm movements. We define a class which contains an integer array for holding integer value of control messages and a String array for description of the control messages. They will be passed to SensorProperty as parameters during construction.

```

public interface NXTHumanoidControl {
    public static final int MOVE_FORWARD = 0;
    public static final int MOVE_BACKWARD = 1;
    public static final int STOP_MOVING = 2;
    public static final int MOVE_ARM = 3;
    public static final int STOP_ARM = 4;

    public static final int[] control = new int[]{
        MOVE_FORWARD,
        MOVE_BACKWARD,
        STOP_MOVING,
        MOVE_ARM,
        STOP_ARM};

    public static final String MOVE_FORWARD_DESC = "Move Forward";
    public static final String MOVE_BACKWARD_DESC = "Move Backward";
    public static final String STOP_MOVING_DESC = "Stop Moving";
    public static final String MOVE_ARM_DESC = "Move Arm";
    public static final String STOP_ARM_DESC = "Stop Arm";

    public static final String[] controlDesc = new String[]{
        MOVE_FORWARD_DESC,
        MOVE_BACKWARD_DESC,
        STOP_MOVING_DESC,
        MOVE_ARM_DESC,
        STOP_ARM_DESC};
}

```

Defining SensorData

Different type of sensor gives output with different characteristics and format. To allow applications decoding data from different sensors effectively, each type of sensor are required to have a class extending `com.anabas.sensorgrid.classification.senordata.SensorData` which defines the properties of data the sensor gives.

Here are classes for some of the NXT robot sensors.

NXTSensorData

```
// all NXT sensor data class extends this class
```

```

public abstract class NXTSensorData extends SensorData{

    public NXTSensorData()
    {

    }

    public NXTSensorData(long timestamp)
    {
        super(timestamp);
    }
}

```

NXTCompassData

```

// data class for compass sensor
public class NXTCompassData extends NXTSensorData{

    // degree of compass
    private int data = 0;

    public NXTCompassData()
    {

    }

    public NXTCompassData(long timestamp, int data)
    {
        super(timestamp);
        this.data = data;
    }

    public int getData()
    {
        return data;
    }
}

```

NXTTemperatureData

```

// data class for temperature sensors
public class NXTTemperatureData extends NXTSensorData {

    // temperature in degree celcius
    private float data = 0.0f;

    public NXTTemperatureData()
    {

    }

    public NXTTemperatureData(long timestamp, float data) {
        super(timestamp);
        this.data = data;
    }

    public float getData() {
        return data;
    }
}

```

```

    }
}

```

NXTRobotData

```

import java.util.Vector;

// this class holds data of all robot sensors as a single entity
public class NXTRobotData extends SensorData{

    private Vector<NXTSensorData> sensorDataList = new
Vector<NXTSensorData>();

    public NXTRobotData()
    {

    }

    public NXTRobotData(long timestamp)
    {
        super(timestamp);
    }

    public void addSensorData(NXTSensorData data)
    {
        sensorDataList.add(data);
    }

    public Vector<NXTSensorData> getSensorDataList()
    {
        return sensorDataList;
    }
}

```

Putting Everything to SensorProperty

Now we have enough definitions to create a SensorProperty object for NXT Humanoid robot. Suppose we want to deploy the robot with the following details:

1. sensorId = "Humanoid"
2. groupId = "demo"
3. sensorType = "NXT ROBOT"
4. sensorTypeId = 0
5. location = "US"
6. historical = true
7. Bluetooth address = 00165302ea7c
8. Robot Type = "Humanoid"
9. Robot attached with touch, sound, light and ultrasonic sensors

The following code should create a SensorProperty object with these details:

```

RobotUserDefinedProperty robotProperty = new RobotUserDefinedProperty(
    "00165302ea7c",
    RobotConstants.HUMANOID,
    RobotConstants.TOUCH,

```



```

RobotConstants.SOUND,
RobotConstants.LIGHT,
RobotConstants.ULTRASONIC
);

SensorProperty property = new SensorProperty(
    "Humanoid",
    "demo",
    "NXT ROBOT",
    0,
    "US",
    true,
    NXTHumanoidControl.control,
    NXTHumanoidControl.controlDesc,
    robotProperty
);

```

A.4.2 Start Connection

To deploy a sensor, the corresponding SCP has to instantiate a `SensorAdapter` object which notifies SCGMMS for its presence and data publishing. The constructor of `SensorAdapter` takes three parameters:

Table 9-2 A sensor adapter object

SensorAdapter	
Parameter:	Description
SensorPolicy	Defines the policy of the sensor. It contains a <code>SensorProperty</code> object (described in the previous subsection) which actually defines all sensor properties
SensorGridControlListener	An interface for listening to control messages from applications
SensorAdapterListener	An interface for listening to application specific events, such as connection loss

To make connection with SCGMMS, the SCP has to create a Java class which implements the `SensorGridControlListener` and `SensorAdapterListener` interface. Details on their usage will be discussed in later subsections.

To initiate the communication between SCP and SCGMMS, creates a `SensorAdapter` object like the following sample code:

```

public class RobotClient implements SensorGridControlListener,
SensorAdapterListener {

    SensorAdapter m_sensorAdapter;

    public void RobotClient(SensorProperty robotProperty)
    {
        // suppose robotProperty is already defined
        SensorPolicy sPolicy = new SensorPolicy(robotProperty)

```

```

        //Start connection
        m_sensorAdapter = new SensorAdapter(sPolicy, this, this);
    }
}

public void handleSensorControl(int ctrl) {}

public void handleSensorControl(int ctrl, Serializable[] parameters) {}

public void handleSensorStopRequest() {}

public void handleSensorConnectionLoss() {}

```

A.4.3 Publish Data

SCP is responsible for collecting data from the sensor, and then publishes it through Sensor Adapter. Sensor Adapter in turn forwards the data to SCGMMS and finally to all applications that need the sensor according to their UDOP requirements.

It is SCP's responsibility to read raw data from the sensor and serialize it into any class which extends SensorData so that they can be interpreted by SCGMMS. The serialized data can then be published through SensorAdapter, which is demonstrated by following sample code:

```

// a class extending SensorData defined by sensor developer
CustomData customData;

/* some code to read raw data from sensor, and put it into customData
 * ...
 * end
 */

// publish the data
m_sensorAdapter.publishData(customData);

```

A.4.4 Receive Control Messages

To receive control messages from applications, SCP should contain a class which implements SensorGridControlListener interface. Two callback functions will be invoked automatically upon arrival of control messages.

For control messages without parameters, the following callback function is used:

```

public void handleSensorControl(int ctrl) {
    // some actions to handle control messages
}

```

Sometimes control messages can be associated with parameters. In this case the following callback function is used:

```

public void handleSensorControl(int ctrl, Serializable[] parameters) {}

```

The type of control message received depends on how they are sent from applications. It is a good idea to define these messages in common libraries which will be used by both sensor and application developers.

A.4.5 Listen to Program Specific Instructions

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 `SensorAdapterListener` interface. Two callback functions will be invoked upon arrival of these events:

This callback function allows applications to stop the SCP

```
public void handleSensorStopRequest() {
    // close the SCP. Release resources
}

public void handleSensorConnectionLoss() {
    // this sensor is not managed by SCGMMS anymore of unexpected
    disconnection. Close the SCP and release resources
}
```

A.5 Sample Program

This section shows a sample program which reads data from a NXT Lego Robot and uses SSAL to connect the device with SCGMMS.

```
import java.io.*;

import cgl.hpsearch.core.policies.SensorPolicy;

import com.anabas.sensorgrid.sensor.SensorGridControlListener;
import com.anabas.sensorgrid.classification.SensorProperty;
import com.anabas.sensorgrid.classification.predefined.PredefinedType;
import com.anabas.sensorgrid.classification.predefined.robot.RobotUserDefinedProperty;
import com.anabas.sensorgrid.classification.predefined.robot.RobotConstants;
import com.anabas.sensorgrid.classification.sensordata.NXTSensorData;
import com.anabas.sensorgrid.classification.sensordata.NXTRobotData;

import com.anabas.sensor.client.WatchDog;
import com.anabas.sensor.sensoradapter.SensorAdapter;
import com.anabas.sensor.sensoradapter.SensorAdapterListener;

public class RobotClient implements SensorGridControlListener,
    SensorAdapterListener {

    private SensorAdapter m_sensorAdapter;

    private SensorPolicy m_sensorPolicy;

    private Robot robot;
    private RobotDataGenerator dataGenerator;
```

```

private boolean isVirtual;

private WatchDog watchDog;

public RobotClient(SensorPolicy sPolicy) {
    this.m_sensorPolicy = sPolicy;

    //Start connection
    m_sensorAdapter = new SensorAdapter(sPolicy, this, this);

    RobotUserDefinedProperty userDefinedProp =
(RobotUserDefinedProperty)m_sensorPolicy.getSensorProperty().getUserDef
inedProp();
    String btAddress = userDefinedProp.getBtAddress();
    int robotType = userDefinedProp.getRobotType();
    int[] robotSensors = userDefinedProp.getPorts();

    this.isVirtual = btAddress.equalsIgnoreCase("Virtual");

    if( !isVirtual ) {
        // change icommand.properties
        ICommandPropertyManager.changeFile(btAddress);
    }

    try {
        robot = new Robot(isVirtual, robotType, robotSensors);
    } catch (RuntimeException e) {
        close();
    }
    dataGenerator = new RobotDataGenerator();
    dataGenerator.start();

    watchDog = new WatchDog(60000) {

        public void timeoutAction() {
            close();
        }
    };

    watchDog.start();
}

private class RobotDataGenerator extends Thread {
    boolean isDestroyed = false;
    private String[] fakeLatLon;

    public void run() {
        while(!isDestroyed) {
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                // ignore
            }

            robot.keepAlive();
        }
    }
}

```

```

        NXTSensorData[] sensorData = robot.sensorData();
        NXTRobotData robotData = new
NXTRobotData(System.currentTimeMillis());
        for( int i = 0; i < sensorData.length; i++ ) {
            robotData.addSensorData(sensorData[i]);
        }
        m_sensorAdapter.publishData(robotData);
        watchDog.refresh();
    }

    public void destroy() {
        isDestroyed = true;
    }
}

public void close() {
    if( dataGenerator != null ) {
        dataGenerator.destroy();
        dataGenerator = null;
    }

    if (robot != null) {
        robot.stopConnection();
        robot = null;
    }

    if( m_sensorAdapter != null ) {
        m_sensorAdapter.close();
        m_sensorAdapter = null;
    }

    if( watchDog != null ) {
        watchDog.destroy();
        watchDog = null;
    }
}

public void handleSensorControl(int ctrl) {
    robot.handleSensorControl(ctrl);
}

public void handleSensorControl(int ctrl, Serializable[] parameters){}

public void handleSensorStopRequest() {
    close();
}

public void handleSensorConnectionLoss() {
    close();
}

public static void main(String[] args) {
    RobotClient myRobotClient = null;
    String btAddress;
    int robotType;

```

```

int[] ports = new int[4];

if( args.length != 9 ) {
    System.out.println("Usage: RobotClient <sensorId> <groupId>
<location> <btAddress> <robotType> <port1> <port2> <port3> <port4>");
    System.exit(-1);
}

try {
    btAddress = args[3];
    robotType = Integer.valueOf(args[4]);
    ports[0] = Integer.valueOf(args[5]);
    ports[1] = Integer.valueOf(args[6]);
    ports[2] = Integer.valueOf(args[7]);
    ports[3] = Integer.valueOf(args[8]);

    if( robotType != RobotConstants.HUMANOID && robotType !=
RobotConstants.TRIBOT ) {
        System.out.println("Robot type is not supported!");
        System.exit(-1);
    }

    for( int i = 0; i < ports.length; i++ ) {
        if( ports[i] != RobotConstants.NONE &&
            ports[i] != RobotConstants.LIGHT &&
            ports[i] != RobotConstants.TOUCH &&
            ports[i] != RobotConstants.SOUND &&
            ports[i] != RobotConstants.ULTRASONIC &&
            ports[i] != RobotConstants.COMPASS &&
            ports[i] != RobotConstants.TEMPERATURE &&
            ports[i] != RobotConstants.GYRO ) {
            System.out.println("Robot sensor type is not supported!");
            System.exit(-1);
        }
    }

    SensorProperty sProp =
PredefinedType.getRobotSensorProperty(args[0], args[1], args[2],
btAddress, robotType, ports[0], ports[1], ports[2], ports[3]);
    SensorPolicy sPolicy = new SensorPolicy(sProp);
    myRobotClient = new RobotClient(sPolicy);

    while( true ) {
        try {
            Thread.sleep(60000);
        } catch (InterruptedException e) {
            // ignore
        }
    }
} catch( NumberFormatException e ) {
    System.out.println("Robot type and all ports must be integer!");
    System.exit(-1);
}
}
}

```

A.6 Deployment

The current implementation requires sensor developers to place the SCP inside Grid Builder package. Since Grid Builder runs on Windows, your SCP should also run on Windows. Please follow the instructions below to package your SCP.

1. Package you SCP into a single Jar file using some Java packaging tools such as Apache Ant.
2. Get the Grid Builder package and extract into a location which does not have any space characters in its full path (e.g. C:\GBPackage). From now on we use <installation directory> to represent this path

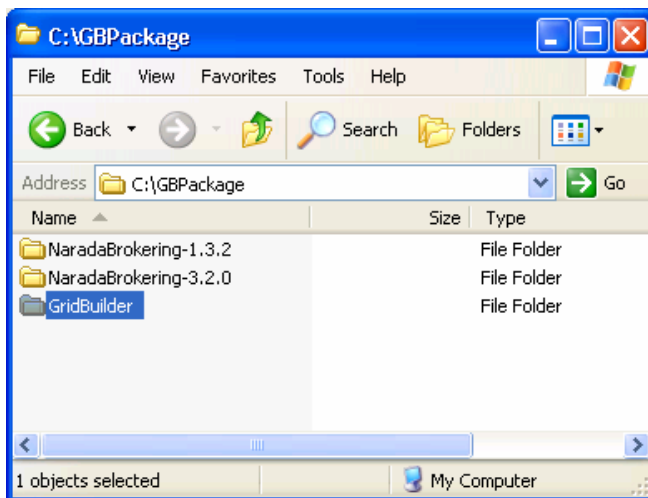


Figure 9-2 Screenshot of Grid Builder deployment

3. In the extracted package, put the SCP's jar file to a directory (e.g. <installation directory>\GridBuilder\scp\). Suppose your SCP's jar file is named "CustomSensor.jar", you file should be placed here:

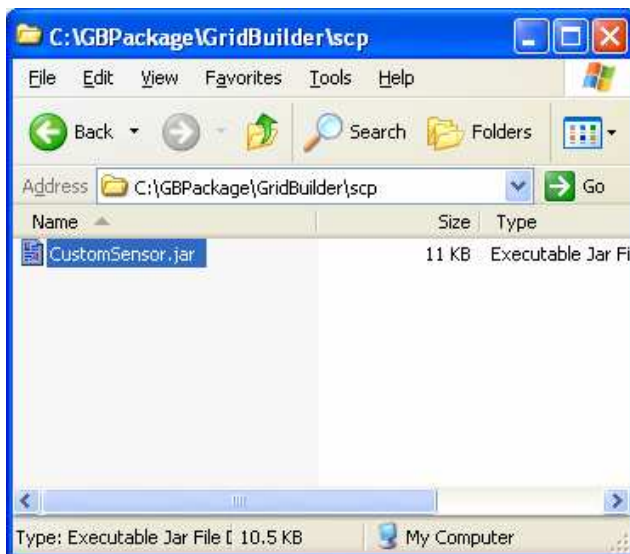


Figure 9-3 Screenshot of Sensor Client Program (SCP) extration

4. Go to <installation directory>\GridBuilder\bin. Create a .bat file with the following format. Remember to replace the name of main class with the main class of your SCP.

```
@echo off

rem USED To start Broker Service Adapter for creating and managing
rem Brokers
rem Usage:
rem runBrokerServiceAdapter --uuid=<UUID> [--resourceGroup=<GROUP>]

set cp=
CALL setEnv.bat

for %%i in ("%HPSEARCH_HOME%\lib\sensor\*.jar") do call cpappend.bat
%%i
set CP=%CP%;"%HPSEARCH_HOME%\scp\CustomSensor.jar"
set LIBPATH=%HPSEARCH_HOME%\lib\sensor\native
set JAVA_LIB=-Djava.library.path=%LIBPATH%

java "%JAVA_LIB%" -classpath %CP% customsensor.RobotClient
```

The directory of your SCP

Main class of your SCP

5. In <installation directory>\GridBuilder\conf\mgmtSystem.conf, setup the NB address of where the SCGMMS server is located. Here is the section in the file where the address is located:

```
# -----
# Config Entries for Service Adapter
# -----

ServiceAdapter.NumOfMessagingNodes=1
ServiceAdapter.MessagingNode_1=192.168.1.51
ServiceAdapter.RegistryUDPPort_1=65050
```

NB address of SCGMMS server

6. Deploy the sensor by executing the .bat file

Appendix B - User Guide for Sensor-Centric Application Developers

Below shows a sample program which uses the Application API:

Application API allows any third party application to connect and utilize functions provided by SCGMMS. An application can do the following through Application API:

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

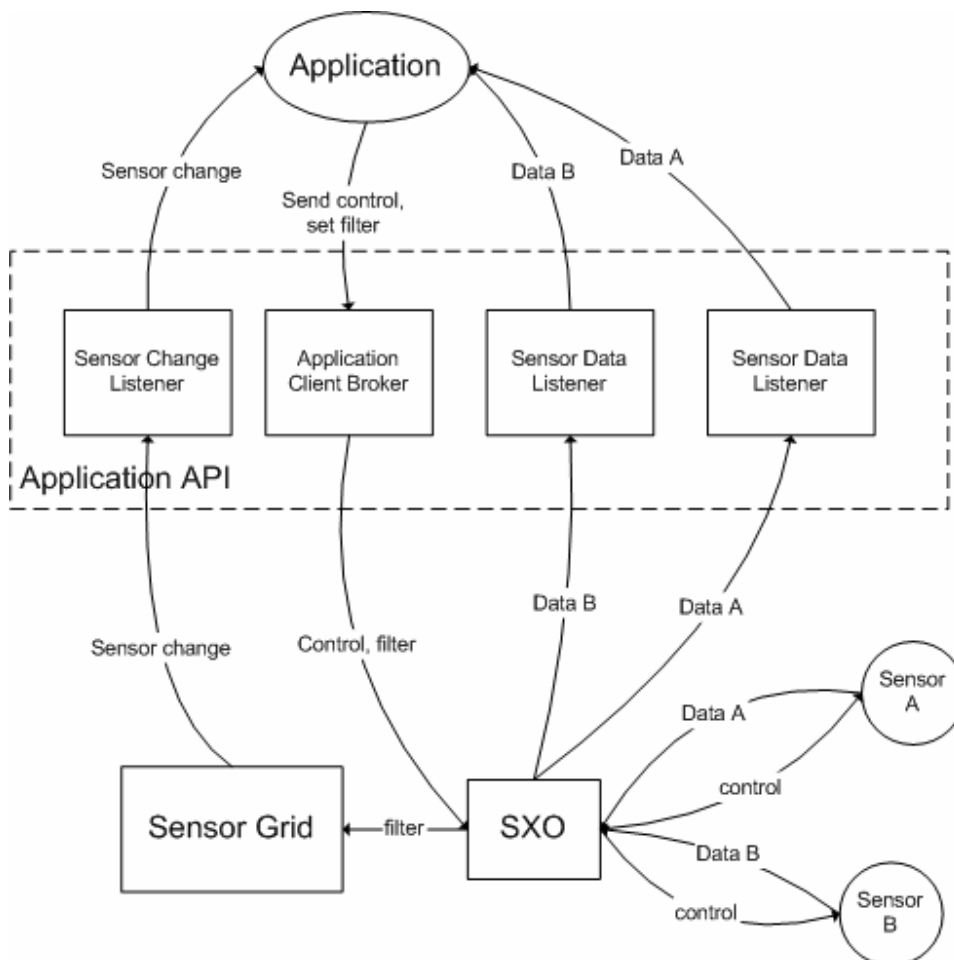


Figure 9-4 Overview of Application API

B.1 System Requirements

The Application API is written in Java. We recommend using Java SE 5 JDK for application development using the Application API.

Here is the recommended System Requirement for PC which runs application developed through the Application API

- Pentium IV 3.0 GHz Processor or above
- 512 MB RAM
- Sun Microsystems Java Runtime Environment 1.5.0 or above

B.2 Libraries

The Application API has a single library file - *SensorGridBroker.jar*. It includes all class files necessary for communication with SCGMMS. To use it as external library, make sure that the file is in the CLASSPATH during compilation.

B.3 Detailed Descriptions

This section provides detailed descriptions on how to use different classes in the API to retrieve information from SCGMMS.

B.3.1 Make Connection

First of all, create a Java class which implements the *SensorGridConnectionLossListener* and *ClientGridChangeListener* interface for detecting connection loss with SCGMMS and receiving sensor change notifications (will be discussed in next subsection).

To initiate the communication between your application and SCGMMS, creates an *ClientGridBroker* object, providing the host and port of a SCGMMS server as parameters. (For instructions on setting up a SCGMMS server please refer to Appendix C -

User Guide for System Administrator). Below shows some sample code to initiate connection with a SCGMMS server.

```
class SampleApplication implements SensorGridConnectionLossListener,
ClientGridChangeListener
{
    // broker for communication with SCGMMS
    ClientGridBroker m_gridBroker;

    public SampleApplication()
    {
        // host and port of SCGMMS
        String sensorGridHost = "64.151.140.118";
        String sensorGridPort = "3035";

        // instantiates the broker
        m_gridBroker = new ClientGridBroker(sensorGridHost, sensorGridPort,
this, this);
    }

    public void handleSensorGridConnectionLoss()
    {
        // some code to handle connection loss, e.g. reconnection
    }
}
```

B.3.2 Sensor Change Notification

After the application is successfully connected to SCGMMS, it will be notified for a list of sensors available through the `handleClientInit()` callback function in `ClientGridChangeListener` interface. Each sensor can be identified by a unique system-generated `String id` and associated with a `SensorGridResource` object. This object contains all the properties of the sensor and will be discussed in detail in the next subsection.

After getting the initial list of sensors, each subsequent change will be notified through the `handleClientChange()` callback function in `ClientGridChangeListener` interface. A parameterized `HashMap<String, SensorGridResource>` is passed as parameter of the callback function, which maps sensor id to resource. There are several reasons which cause changes in sensor properties, including:

1. new sensors are deployed
2. some sensors are disconnected (for cause or accidentally)
3. application changes the filtering criteria

The third reason will be discussed later in this section.

Below shows sample code of `handleClientInit()` and `handleClientChange()` callback functions.

```

public void handleClientInit(HashMap<String, SensorGridResource> sinfo)
{
    // create an iterator from sinfo
    Iterator<String> it = sinfo.keySet().iterator();

    while (it.hasNext()) {
        // unique system generated sensor id
        String sensorID = it.next();

        // sensor resource
        SensorGridResource resource = sinfo.get(sensorID);

        // get sensor policy
        SensorPolicy policy = (SensorPolicy) resource.getPolicy();
    };
}

public void handleClientChange(HashMap<String, SensorGridResource>
sinfo) {
    // create an iterator from sinfo
    Iterator<String> it = sinfo.keySet().iterator();

    while (it.hasNext()) {
        // unique system generated sensor id
        String sensorID = it.next();

        // sensor resource
        SensorGridResource resource = sinfo.get(sensorID);

        // online, offline status of sensor
        Short status = resource.getStatus();

        // sensor policy
        SensorPolicy policy = (SensorPolicy) resource.getPolicy();
    }
}

```

B.3.3 Process Sensor Policy

After getting the SensorGridResource of sensors from ClientGridChangeListener, the application can access various properties of the sensor through this object.

Table 9-3 A sensor grid resource interface

SensorGridResource	
Interface:	Description
Policy getPolicy()	Get a Policy object which describes the properties of the sensor
short getStatus()	Get the online (represented by SensorStatus.ONLINE) and offline (represented by SensorStatus.OFFLINE) status of the sensor

After getting Policy, the application should cast it to SensorPolicy using the SensorPolicy interface as shown in Table 9-4.

Table 9-4 A sensor policy interface

SensorPolicy	
Interface:	Description
SensorProperty getSensorProperty()	Get a SensorProperty object which describes the properties of the sensor

A SensorProperty object contains metadata of a sensor, which can be extracted using the SensorProperty interface depicted in Table 9-5.

Table 9-5 A sensor property interface

SensorProperty	
Interface:	Description
String getId()	A string which helps identifying the sensor. Different from the unique system-generated id
String getGroupId()	A string which identifies the name of logical group which the sensor belongs to
String getSensorType()	A string which represents the type of sensor. A list of predefined types are defined as static variables in class PredefinedType (e.g. PredefinedType.VIDEO)
int getSensorTypeId()	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
String getLocation()	A string which represents the geo-spatial location of the sensor. A list of predefined locations can be found in class PredefinedLocation
boolean isHistorical()	Whether sensor data has time inter-dependence with one another
int[] getSensorControl()	An integer array which identifies control messages understood by a sensor
String[] getControlDescription()	Textual description of control messages
UserDefinedProperty getUserDefinedProp()	Sensor developers can create classes which extends UserDefinedProperty and put sensor-specific properties inside the class

B.3.4 Subscribe Sensor Data

After the application is aware of the presence of a sensor through ClientGridChangeListener, it can subscribe or unsubscribe to the data stream of this sensor. The application has to create a class which implements the ClientGridDataListener interface and pass it to ClientGridBroker.subscribeSensorData() with the data listener and sensor id as parameters. handleSensorData() will be invoked whenever data arrives. Different sensor types have to define its own data class by extending SensorData. This class should encapsulate the specific data structures for different sensors.

Here is the sample code:

```
// subscribe to data of sensor123
DataMonitor dataMonitor = new DataMonitor("sensor123");
m_gridBroker.subscribeSensorData("sensor123", dataMonitor);

// unsubscribe to data of sensor123
m_gridBroker.unsubscribeSensorData("sensor123", dataMonitor);

// data listener class
```

```

private class DataMonitor extends Thread implements
ClientGridDataListener {
    boolean isDestroyed = false;
    private String id;
    private GenericSensorControl ctrl = new GenericSensorControl();

    public DataMonitor(String id) {
        this.id = id;
    }

    public void run() {
        while(!isDestroyed) {
            try {
                Thread.sleep(10000);
            } catch (InterruptedException e) {
                // ignore
            }
        }
    }

    public void destroy() {
        isDestroyed = true;
    }

    public void handleSensorData(SensorData data) {
        if( data instanceof GpsData ) {
            // data from GPS
        } else if( data instanceof RfidSensorStrengthData ) {
            // data from RFID tag
        }
    }
}

```

B.3.5 Filtering

An application does not always want information from all sensors. It is allowed to filter away those sensors which do not match some criteria. The criteria are defined by a `SensorFilter` object. A `SensorFilter` is composed of a set of properties defined in `SensorProperty` connected with Boolean “and” or “or” operators. Given that a list of sensor properties in a sensor filter are connected together with the “and” operator, only sensors which have properties with exact match in string comparison with ALL the properties defined in the filter should get through. Similarly sensors which have properties with 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 SAID 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);

```

After the SensorFilter object is created, send it to SCGMMS with the following sample code:

```

m_gridBroker.setFilter(filter);

```

After SCGMMS receives the request, it examines the filter and checks it with the current list of sensors for the application. It then notifies the application through handleSensorChange() of ClientGridChangeListener by setting the properties of sensors which do not fulfill the filtering criteria as offline and those fulfilling the criteria as online.

B.3.6 Send Control

An application is able to send control to a particular sensor identified by its sensor id. Each control message is just an integer and its meaning is defined in SensorProperty. Here is the sample code for sending control message.

```

m_gridBroker.sendControl("sensor123", 1);

```

B.4 Sample Application

```

package com.anabas.meeting.test;

import java.util.*;
import java.io.*;

import com.anabas.sensorgrid.client.ClientGridBroker;
import com.anabas.sensorgrid.client.ClientGridChangeListener;
import com.anabas.sensorgrid.client.ClientGridDataListener;
import com.anabas.sensorgrid.client.ClientGridSensorStatus;
//import com.anabas.sensorgrid.client.ClientGridResource;

import
com.anabas.sensorgrid.classification.sensorControl.GenericSensorControl;
import com.anabas.sensorgrid.classification.sensordata.*;

import cgl.hpsearch.core.policies.SensorPolicy;
import com.anabas.sensorgrid.classification.SensorProperty;

import com.anabas.sensorgrid.session.sharedlet.SGResource;

import com.anabas.util.misc.LogManager;

```



```

public class MeetingTest implements ClientGridChangeListener {
    ClientGridBroker m_broker;
    HashSet<String> m_onlineSensors;
    HashMap<String, DataMonitor> m_sensorID2Monitor;
    HashMap<String, SensorProperty> m_sensorID2Property;
    Object m_lock = new Object();

    public MeetingTest(String hostname, String port) {
        m_broker = new ClientGridBroker(this, hostname, port);
    }

    public void handleSensorInit(HashMap<String, SGResource>
sensorInitInfo) {
        synchronized( m_lock ) {
            System.out.println("\n\n\nNumber of sensors: " +
sensorInitInfo.size() + "\n\n\n");

            m_onlineSensors = new HashSet<String>();
            m_sensorID2Monitor = new HashMap<String, DataMonitor>();
            m_sensorID2Property = new HashMap<String, SensorProperty>();

            Iterator<String> it = sensorInitInfo.keySet().iterator();
            while( it.hasNext() ) {
                String sensorID = it.next();
                m_onlineSensors.add(sensorID);

                SensorPolicy policy =
(SensorPolicy)sensorInitInfo.get(sensorID).getPolicy();

                if( policy == null ) {
                    System.out.println("\n\n\nPolicy is null!!!\n\n\n");
                } else {
                    System.out.println("\n\n\nPolicy is not null!!!\n\n\n");
                }
                m_sensorID2Property.put( sensorID,
((SensorPolicy)sensorInitInfo.get(sensorID).getPolicy()).getSensorPrope
rty() );

                DataMonitor monitor = new DataMonitor(sensorID);
                monitor.start();

                m_sensorID2Monitor.put( sensorID, monitor );
                m_broker.subscribeSensorData(sensorID, monitor);
            }
        }
    }

    public void handleSensorChange(HashMap<String, SGResource>
sensorChangeInfo) {
        synchronized( m_lock ) {
            Iterator<String> it = sensorChangeInfo.keySet().iterator();

            while( it.hasNext() ) {
                String sensorID = it.next();
                SGResource resource = sensorChangeInfo.get(sensorID);
                Short status = resource.getStatus();
            }
        }
    }
}

```

```

        if( status == ClientGridSensorStatus.ONLINE ) {
            System.out.println("\n\n\nHERE!!!\n\n\n");

            m_onlineSensors.add(sensorID);
            SensorPolicy policy = (SensorPolicy)resource.getPolicy();

            if( policy == null ) {
                System.out.println("\n\n\nPolicy is null!!!\n\n\n");
            } else {
                System.out.println("\n\n\nPolicy is not null!!!\n\n\n");
            }

            m_sensorID2Property.put( sensorID,
((SensorPolicy)resource.getPolicy()).getSensorProperty() );

            DataMonitor monitor = new DataMonitor(sensorID);
            monitor.start();

            m_sensorID2Monitor.put( sensorID, monitor );
            m_broker.subscribeSensorData(sensorID, monitor);
        } else {
            m_onlineSensors.remove(sensorID);
            m_sensorID2Property.remove(sensorID);

            DataMonitor monitor = m_sensorID2Monitor.remove(sensorID);
            m_broker.unsubscribeSensorData(sensorID, monitor);

            monitor.destroy();
        }
    }
}

public static void main(String[] args) {
    if(args.length != 2) {
        System.out.println("Usage: java GPSManager <hostname> <port>");
        System.exit(0);
    }

    MeetingTest test = new MeetingTest(args[0], args[1]);
    while (true) {
        try{
            Thread.sleep(10000);
            System.gc();
        } catch (java.lang.InterruptedException e){
            // ignore
        }
    }
}

private class DataMonitor extends Thread implements
ClientGridDataListener {
    boolean isDestroyed = false;
    private String id;
    private GenericSensorControl ctrl = new GenericSensorControl();

```

```

public DataMonitor(String id) {
    this.id = id;
}

    public void run() {
        while(!isDestroyed) {
            try {
                Thread.sleep(10000);
            } catch (InterruptedException e) {
                // ignore
            }
        }
    }

    public void destroy() {
        isDestroyed = true;
    }

    public void handleSensorData(SensorData data) {
        if( data instanceof GpsData ) {
            GpsData gpsData = (GpsData)data;
            String output = "\n\n\nData received, id: " + id + " , data: "
+ gpsData.getLat() + " , " + gpsData.getLng() + " , timeStamp: " +
gpsData.getTimestamp() + "\n\n\n";
            LogManager.log("DataMonitor", output);
        } else if( data instanceof RfidSensorStrengthData ) {
            RfidSensorStrengthData rfidData = (RfidSensorStrengthData)data;
            String output = "\n\n\nData received, id: " + id + " , data: "
+ rfidData.getSignalStrength() + "\n\n\n";
            LogManager.log("DataMonitor", output);
        } else if( data instanceof NXTTouchData ) {
            NXTTouchData robotData= (NXTTouchData)data;
            String output = "\n\n\nData received, id: " + id + " , data: "
+ String.valueOf(robotData.getData()) + "\n\n\n";
            LogManager.log("DataMonitor", output);
        } else if( data instanceof NXTSoundData ) {
            NXTSoundData robotData = (NXTSoundData)data;
            String output = "\n\n\nData received, id: " + id + " , data: "
+ String.valueOf(robotData.getData()) + "\n\n\n";
            LogManager.log("DataMonitor", output);
        } else if( data instanceof NXTUltrasonicData ) {
            NXTUltrasonicData robotData = (NXTUltrasonicData)data;
            String output = "\n\n\nData received, id: " + id + " , data: "
+ String.valueOf(robotData.getData()) + "\n\n\n";
            LogManager.log("DataMonitor", output);
        } else if( data instanceof NXTTemperatureData ) {
            NXTTemperatureData robotData = (NXTTemperatureData)data;
            String output = "\n\n\nData received, id: " + id + " , data: "
+ String.valueOf(robotData.getData()) + "\n\n\n";
            LogManager.log("DataMonitor", output);
        } else if( data instanceof NXTCompassData ) {
            NXTCompassData robotData = (NXTCompassData)data;
            String output = "\n\n\nData received, id: " + id + " , data: "
+ String.valueOf(robotData.getData()) + "\n\n\n";
            LogManager.log("DataMonitor", output);
        } else if ( data instanceof NXTLightData ) {
            NXTLightData robotData = (NXTLightData)data;

```

```

        String output = "\n\n\nData received, id: " + id + " , data: "
+ String.valueOf(robotData.getData()) + "\n\n\n";
        LogManager.log("DataMonitor", output);
    }

    //m_broker.sendControl(id, ctrl);
}
}
}

```

Appendix C - User Guide for System Administrator

In SCGMMS, a perpetual session server known as Sensor Grid (SG) has to be up and running all the time to communicate with sensors, Grid Builder and applications. This guide will show you how to setup a SG server.

C.1 System Requirements

Please make sure that all minimum are met in your operating environment before installing and using the SG server. Not meeting all the minimum requirements may cause undesired system behavior that includes inoperable system or affects or impairs conferencing sessions.

C.1.1 SG Server Requirements

Recommended Server Requirements

- Fedora FC4
- Pentium IV 3.2 GHz processor or above
- 1 GB RAM
- 210 GB disk space
- Access to SMTP mail server

C.2 Server Installation Preparations

Before you start the installation process

- (1) Ensure that your network configuration is setup properly.
- (2) Ensure that the proper ports are open for installation, and
- (3) Ensure that you have root privilege on the Linux server

C.2.1 Verifying Your Network Configuration

Here we need to verify certain network configuration is correct. Follow these simple steps to ensure a successful installation.

Make sure that the "hostname" command works by typing

```
> hostname -i
```

This command will report your computer's IP address if it is working properly. If it reports nothing, 127.0.0.1, or some errors then you will need to add a hostname entry.

Perform the following modification to /etc/hosts to ensure the network is configured for proper operation.

To modify /etc/hosts, add the hostname and IP address of the installation machine. For example, if your computer is called "elearningconferencing.me.com" and its IP address is 65.112.117.218, then you should add the line at the beginning of the /etc/hosts file:

```
65.112.117.218          elearningconferencing.me.com
```

Please restart the server for the settings to take effect

C.2.2 Ensuring A Set of Open Ports

The SG server requires several ports to be open. We recommend you use a dedicated machine. The following is a list of ports which have to be opened:

25050, 3035, 80, 5060, ALL UDP ports

C.2.3 Firewall Settings

To ensure that the above ports are exposed, you always have to check your firewall settings. In Linux FC4 firewall rules are usually defined by **iptables**. If you are familiar with firewall settings you may skip this section. Instructions below describe how to disable the firewall in FC4.

- (1) Check if any rules are defined in iptables by:
> *iptables -L*
- (2) Change iptables configuration so that the firewall settings are saved on reboot.
You are recommend to backup this file before editing
> *vi /etc/sysconfig/iptables-config*

Find and change the following entries in the file:

```
IPTABLES_SAVE_ON_STOP="yes"  
IPTABLES_SAVE_ON_RESTART="yes"  
IPTABLES_SAVE_COUNTER="yes"
```

- (3) Clear all rules of iptables
> *iptables -F*

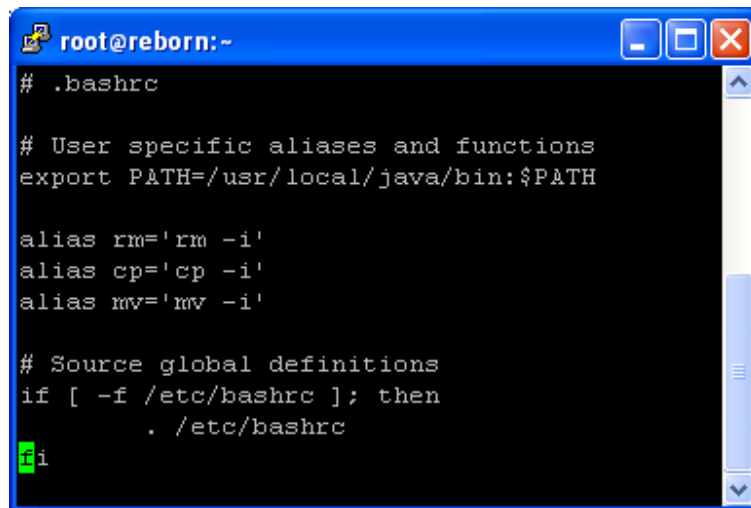
C.2.4 Java Virtual Machine

If this is the first time installing SG on your Linux server, please make sure that a JVM is installed. We recommend using JDK 1.5.x. Please follow the instructions below for installing the JVM.

- (1) Download the JDK package from http://202.94.237.244/tools/jdk-1_5_0_06-linux-i586-rpm.bin. For instance, our installation package is called **jdk-1_5_0_06-linux-i586-rpm.bin**
- (2) Change the file to executable mode, and install it

```
> chmod +x jdk-1_5_0_06-linux-i586-rpm.bin  
> ./jdk-1_5_0_06-linux-i586-rpm.bin
```
- (3) Add a soft link **/usr/local/java** to the JDK directory

```
> ln -s /usr/java/jdk1.5.0_06 /usr/local/java
```
- (4) Open a file **~/.bashrc** with VM and add a line **export PATH=/usr/local/java/bin:\$PATH** to the file



```
root@reborn:~  
# ~/.bashrc  
  
# User specific aliases and functions  
export PATH=/usr/local/java/bin:$PATH  
  
alias rm='rm -i'  
alias cp='cp -i'  
alias mv='mv -i'  
  
# Source global definitions  
if [ -f /etc/bashrc ]; then  
    . /etc/bashrc  
fi
```

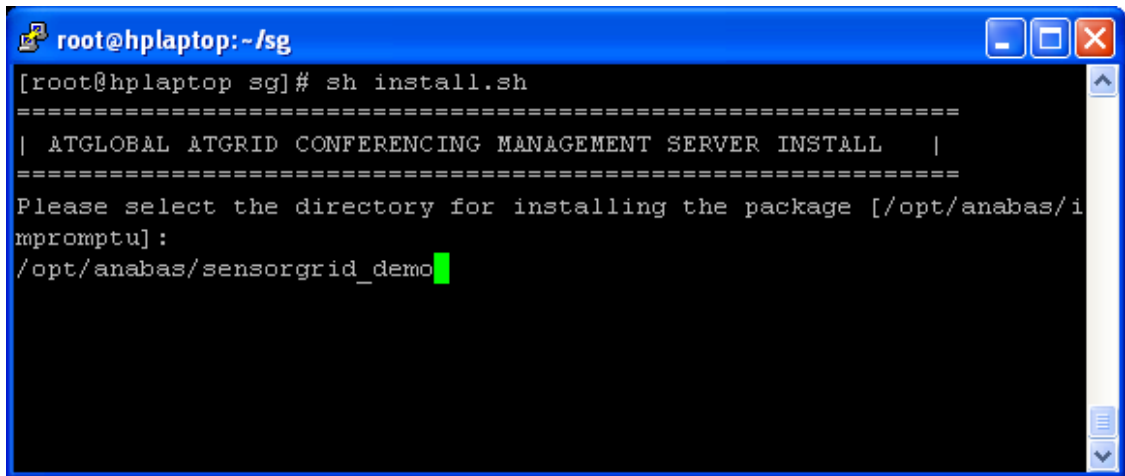
- (5) You have to restart the terminal for the new **PATH** to take effect

C.3 Server Installation

Obtain the installation package. It should be in jar format. Suppose the installation file is named `SGInstaller.jar`. Follow these steps to install it on your server.

You need to have super user privilege to install this package.

- (a) Unjar **SGInstaller.jar** into a temporary directory and run the following commands:
 `> /usr/local/java/bin/jar xvf SGInstaller.jar`
 `> sh install.sh`
- (b) Enter the target installation directory. Suppose we want to install SG to `/opt/anabas/sensorgrid_demo`



The screenshot shows a terminal window titled 'root@hplaptop: - /sg'. The user has run the command `sh install.sh`. The terminal output displays a header: `=====`
`| ATGLOBAL ATGRID CONFERENCING MANAGEMENT SERVER INSTALL |`
`=====`. Below this, it asks: `Please select the directory for installing the package [/opt/anabas/i`. The user has entered `/opt/anabas/sensorgrid_demo` and the prompt is now `mpromptu] :`. A green cursor is visible at the end of the entered path.

The package will be extracted and installed into the directory you have chosen as the installation directory. From now on, we will refer to this directory as <SG HOME>. Toward the end of the installation, you will be asked to configure the most important parameters of the system.

- (c) Please go to the next step if you cannot see this screen. If the hostname of your server is not correct, the wizard will try to help you making it correct. However, it is your responsibility to make sure that the IP address and hostname of your server is correct before the installation. Please refer to the installation preparation section for details.


```
root@hplaptop:/opt/anabas
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_14-b03)
Java HotSpot(TM) Client VM (build 1.5.0_14-b03, mixed mode, sharing)

*****
check if the hostname command is working ...
The hostname command is not working, you need to modify
the /etc/hosts file to activate it.
Do you want me to modify that for you? [y/n]
```

- (d) After a while we proceed to the port configuration section of the installation. To use default ports just press the “enter” key every time a question is asked. We recommend using the default ports. Otherwise, you have to make sure that the ports do not clash with other applications.

```
root@hplaptop:~/sg
=====
| ATGlobal ATGrid Configuration                               |
|                                                             |
| Note: Press the enter key to retain current values.       |
|                                                             |
=====
Grid builder communication server port
-----
[Default: 25050] >

Communication server port
-----
[Default: 3035] >

Web server port
-----
[Default: 80] >

Mail Sender Account:
The e-mail address the system would use for sending e-mails to users
-----
[Default: meeting.notice@anabas.com] >

SMTP Mail Server
-----
[Default: localhost] >

System Alert E-mail:
The e-mail address the system will send alerts to
-----
[Default: root] >
```

- (e) Now we enter the server IP definition section. 3 sets of IP/hostname will be asked. For the first two sets just enter the IP which will be exposed to all applications and sensors, which should be same as the one get by hostname -i. For the third set, it is the address of media server for VOIP module. For now, it doesn't affect the SG server.

```
root@hplaptop: ~/sg
:: Doing Final Preparations For Installation.... ::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

Preparing ..... Please wait for a few minutes
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: Installing system monitors ..... ::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

Detected: linux
Detected: installdir /opt/anabas/sensorgrid_demo
Successfully installed Anabas monitors
Installation Complete!
The IP of the ATGrid host (211.147.225.216):
192.168.1.9
192.168.1.9
The hostname of the ATGrid host (crystalgroup.atgrid.co):
192.168.1.9
192.168.1.9
The IP of the ATGrid Gateway (218.213.238.196):
192.168.1.9
192.168.1.9
The hostname of the ATGrid Gateway (gateway.atgrid.com):
192.168.1.9
192.168.1.9
The IP of the ATGrid Media Server (211.147.225.217):
202.94.237.243
202.94.237.243
The hostname of the ATGrid Media Server (mediaserver.atgrid.com):
202.94.237.243
```

- (f) Please use “supplychain” here and ignore the last warning message

```
root@hplaptop: ~/sg
202.94.237.243
202.94.237.243
Please select between modes (s for supplychain, w for workgroup):
s
s
cat: /opt/anabas/sensorgrid_demo/sipurl.txt: No such file or director
y
You have new mail in /var/spool/mail/root
[root@hplaptop sg]#
```

(g) The installation is completed.

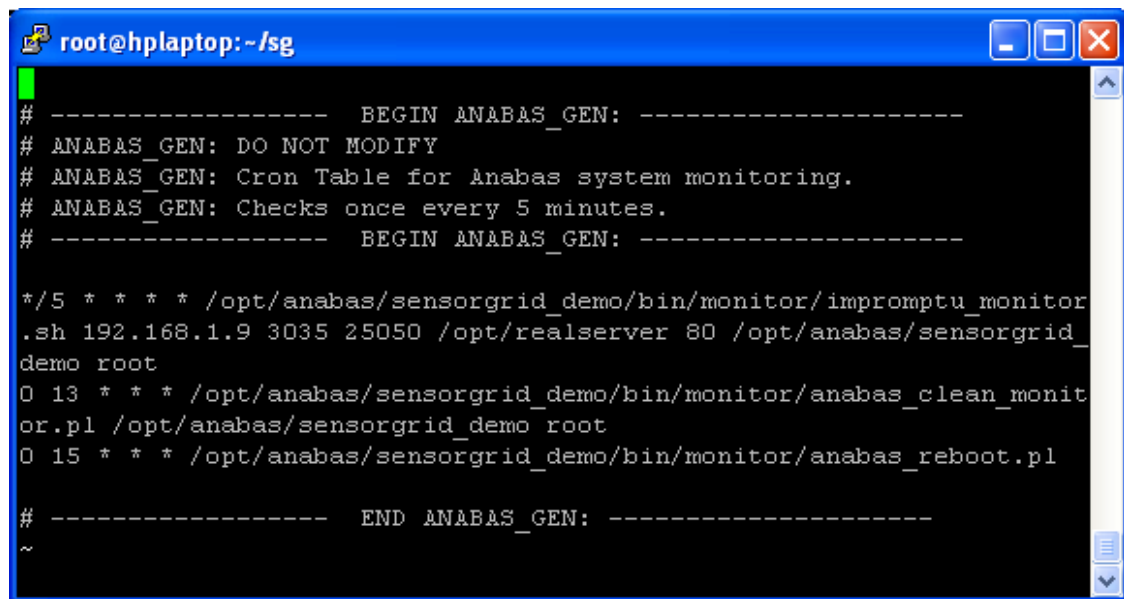
C.4 Server Execution

C.4.1 Starting the packages

After installation, the server will be started automatically by Cronjob.

To check the cronjob file, open the crontab editor with the following command:

> *crontab -e*

A screenshot of a terminal window titled 'root@hplaptop: ~ /sg'. The terminal displays the content of a crontab file. It starts with a header section enclosed in dashed lines, containing instructions: 'BEGIN ANABAS_GEN: DO NOT MODIFY', 'ANABAS_GEN: Cron Table for Anabas system monitoring.', and 'ANABAS_GEN: Checks once every 5 minutes.' Below this, there are three cron entries: '* /5 * * * * /opt/anabas/sensorgrid_demo/bin/monitor/impromptu_monitor.sh 192.168.1.9 3035 25050 /opt/realserver 80 /opt/anabas/sensorgrid_demo root', '0 13 * * * /opt/anabas/sensorgrid_demo/bin/monitor/anabas_clean_monitor.pl /opt/anabas/sensorgrid_demo root', and '0 15 * * * /opt/anabas/sensorgrid_demo/bin/monitor/anabas_reboot.pl'. The file ends with 'END ANABAS_GEN:' and a tilde '~' indicating the end of the file. The terminal window has standard Linux window controls (minimize, maximize, close) in the top right corner.

This table is responsible for starting our server processes at a predefined interval. This table shows 3 entries. The first line means that system will keep checking if SG is running on the server. The second line means that log files will be cleaned and archived every day at 13:00. The third line means that the server will be rebooted at 15:00 every day.

To stop the server, type the following command to kill all processes of the server:

> *killall java*

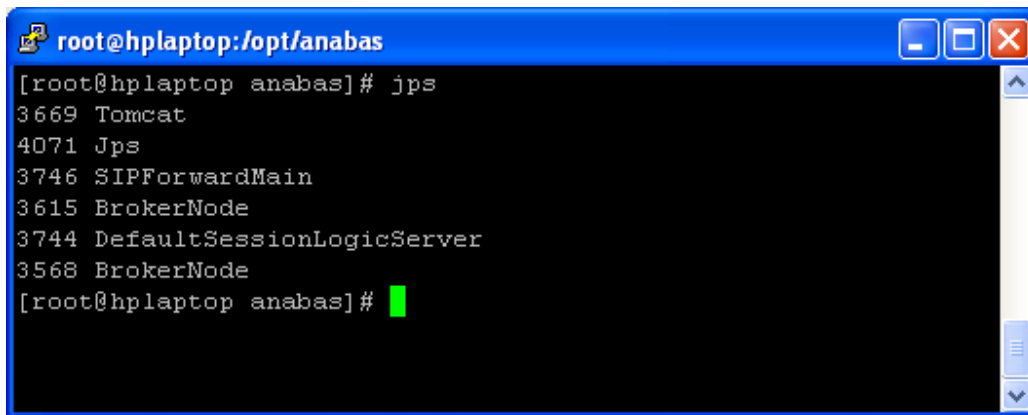
Unless you have commented out the entries in cronjob, the server will start itself automatically for every 5 minutes.

C.4.2 Check the status of server

You can check whether the server is running properly by the following command:

> *jps*

If everything is fine, you should be able to see the following processes running:

A terminal window titled 'root@hplaptop:/opt/anabas' with standard window controls. The terminal shows the output of the 'jps' command, listing several Java processes with their PIDs and names. The prompt is '[root@hplaptop anabas]#' followed by a green cursor.

```
root@hplaptop:/opt/anabas
[root@hplaptop anabas]# jps
3669 Tomcat
4071 Jps
3746 SIPForwardMain
3615 BrokerNode
3744 DefaultSessionLogicServer
3568 BrokerNode
[root@hplaptop anabas]#
```

Now the installation is completed. Sensors, applications and Grid Builder can connect to this server from now on.

Appendix D - User Guide for Sensor Administrator

The role of a sensor administrator is to manage sensor definition and deployment using Grid Builder (GB). Before we start, a brief introduction to GB concept is given below:

D.1 Domain Management

To allow a flexible way to manage sensors, GB administration is arranged hierarchically into **Domains**. Each domain should run on a single PC which manages sensors which are closely related. In each domain, each module is run as a separate process which communicates with one another through NaradaBrokering. Different module has different responsibility. **Fork Daemon** is responsible for starting up different modules as processes. **Bootstrap Service** monitors the health of a domain and the whole domain hierarchy. **Messaging Node** is responsible for intra and inter-domain communications using NaradaBrokering.

Each domain is connected to at most one parent domain and any number of child domains. The hierarchy is maintained by inter-communication between the domains using **heartbeat messages**. The role of a domain is different according to their relative position in the hierarchy. There are three types of domains:

Root Domain

For each domain hierarchy there exists a single **Root Domain**. The Root Domain is responsible for checking whether a Bootstrap Service is running in each directly connected child domain. If not, it notifies the Fork Daemon of that particular domain to start Bootstrap Service by sending a **fork message**. This process is done recursively for each domain along the hierarchy until the Leaf Domains are reached. Root Domain is the starting point of building up the domain hierarchy. No sensors can be deployed in Root Domain.

Sub Domain

A domain which is neither a Root Domain nor a Leaf Domain is a Sub Domain. Each Sub Domain is responsible for checking the Bootstrap Service of its child domains. No sensors can be deployed in Sub Domains.

Leaf Domain

A Leaf Domain does not have any children. Sensors can be deployed from any PC which is accessible from one of the **Leaf Domains**. The current implementation does not allow deployment of sensors on non-leaf domains.

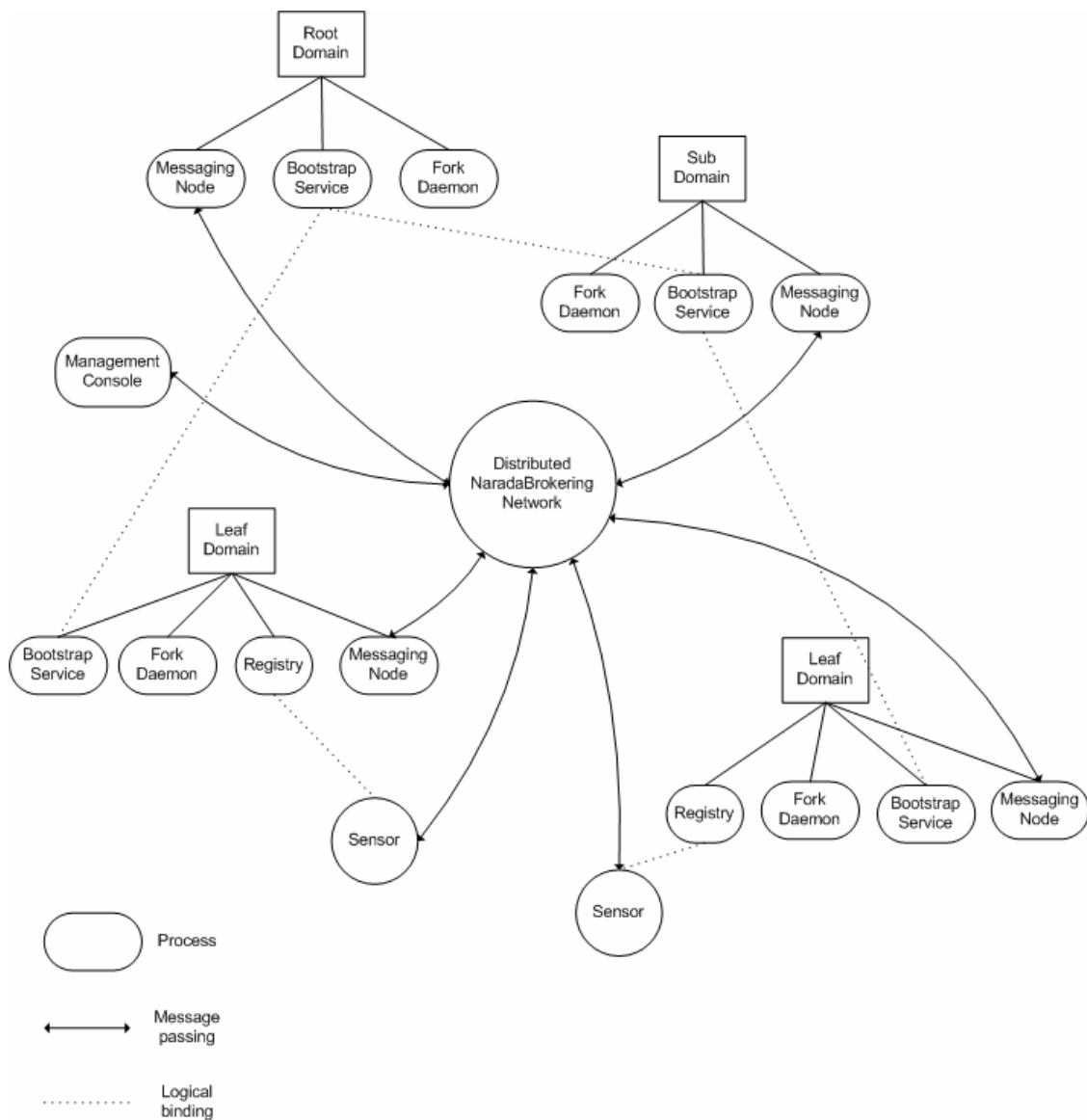


Figure 9-5 GB Domain Management

Figure 9-5 shows a GB domain hierarchy. Each domain consists of several processes. Inter-process communication is done by messaging passing. Through message passing domains and sensors are logically linked together with the use of heartbeat message and in memory hash tables.

D.2 Installation Preparation

D.2.1 System Requirement

Here is the recommend System Requirement for PC of a domain

- Pentium IV 3.0 GHz Processor or above

- 512 MB RAM
- Internet Explorer 6.0+ (requires all Microsoft recommended IE critical patches)
- Windows XP (requires all Microsoft recommended OS critical patches)
- Sun Microsystems Java Runtime Environment 1.5.0 or above

D.2.2 Network Requirement

The current implementation uses UDP protocol for inter-domain communication, including heartbeat and fork messages. To ensure proper functioning of GB, make sure that you fulfill the following network requirements:

- All PCs in the domain hierarchy should be accessible to one another
- At least 4 UDP ports should be open for access from other domains. The port number should match your configurations (see section D.4 for details)
- Port 3035 opened for sensor client programs
- Port 25050 opened for NB communications

D.3 Installation Package

Please get the required zip package and extract it to a location. The full path of the extracted location should NOT contain any space characters (e.g. c:\GBPackage). The installation package comes with 4 modules:

Grid Builder

This is the main package for sensor management.

NaradaBrokering-1.3.2

This package is a NaradaBrokering client used by sensor client programs to communicate with Sensor Grid. It has to be started manually on PCs where sensor client programs are launched.

NaradaBrokering-3.2.0

This package is a NaradaBrokering client used by GB's Messaging Node for communication with sensors and Sensor Grid.

WS-Context

This package contains all elements needed for WS-Context support such as AXIS server and MySQL server.

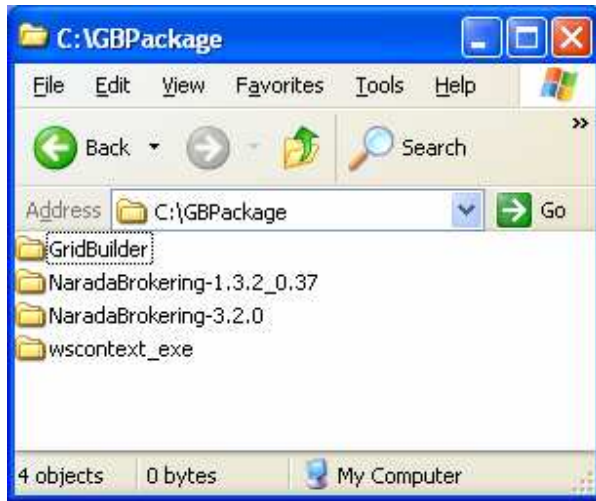


Figure 9-6 GB Package

D.4 Grid Builder Management Console

The Grid Builder Management Console (GBMC) provides a graphical user interface for sensor deployment. From any PC with GB installed, you can view sensors deployed in domains connected by a particular NB network through GBMC.

On the left hand side all domains detected in the NB network are shown with sensors deployed within the domains. On the right hand side various information of sensor is shown, including:

1. Current status – REGISTERED, MANAGED or UNREACHEABLE
2. UUID – unique ID assigned to the sensor
3. Policies – the property of the sensor, such as sensor type, location and user defined properties

Please refer to section D.6 for sample usage.

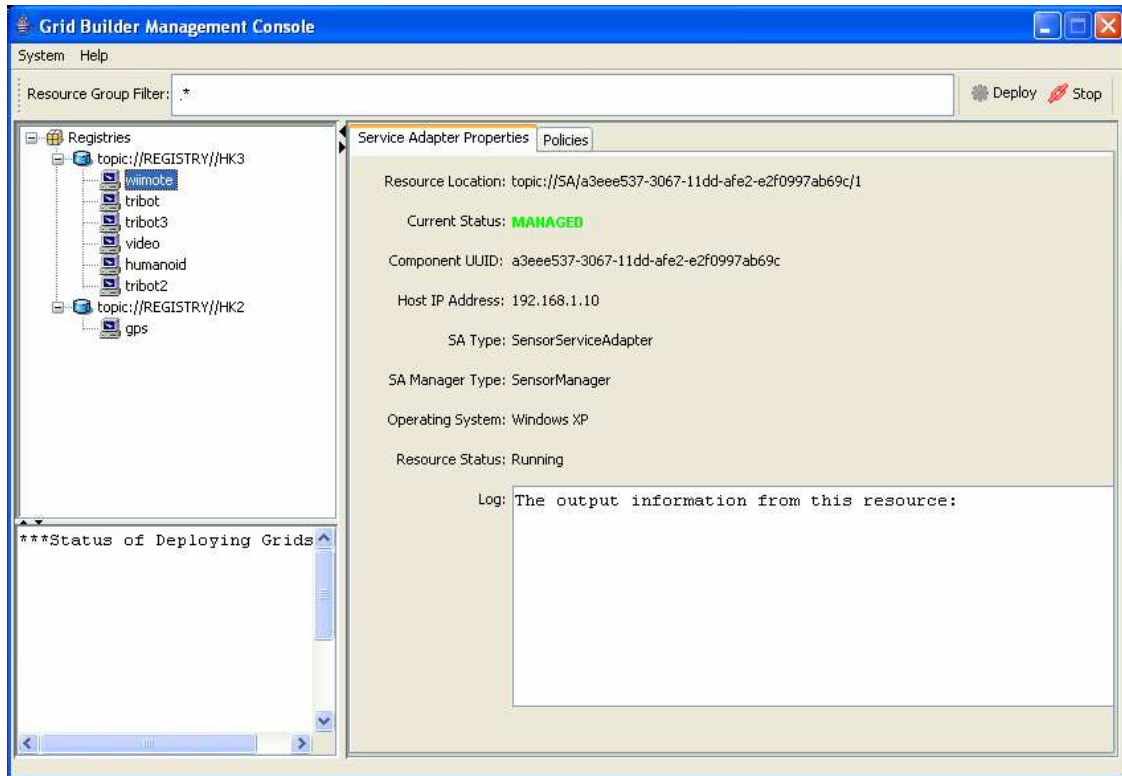


Figure 9-7 GB Management Console

D.5 Configuration Files

For each domain there is a configuration file which defines various parameters of a domain. The following configuration files are present which serve different purposes:

D.5.1 mgmtSystem.conf

The full path of this file is *GridBuilder/conf/mgmtSystem.conf*. This file contains:

1. The unique identifier of the domain
2. The name of the domain
3. Locator of Fork Daemon, Bootstrap Service and Registry of the domain
4. Number of child domains and their corresponding Fork Daemon locators
5. Locator of Messaging Node used for deploying sensors
6. Locator of Messaging Node which the User Management Console connects to
7. Locator of Messaging Node which the Bootstrap UI connects to

The name of a domain follows the domain naming syntax described below:

- Each domain in a hierarchy should have a unique name
- For Root Domain we tag it with keyword ROOT followed by a hyphen and then the word bootstrap.
- For subsequent domains, we put an underscore after the domain name E.g.
ROOT/CGL becomes ROOT_CGL-bootstrap
ROOT/UK/CARDIFF becomes ROOT_UK_CARDIFF-bootstrap etc.

Here are sample mgmtSystem.conf files for two Domains namely HK1 and HK2. HK1 is the ROOT domain.

Configuration file of Domain HK1

```
# -----
# Config Entries for Fork Daemon
# -----
## This string should be unique for different networks
## It is used to uniquely identify a Fork Daemon
ForkDaemon.UniqueString=ATGLOBAL-HK1

# -----
# Config Entries for Root Bootstrap Node
# -----

# The domain of the bootstrap program
ROOT-bootstrap.Level=/
ROOT-bootstrap.ForkProcessLocator=topic://FORKDAEMON/ATGLOBAL-
HK1/:65535

# Number of registered subDomains
ROOT-bootstrap.NumOfRegisteredSubDomains=2

# Domain URI of subDomains and their locations
ROOT-bootstrap.RegisteredSubDomain_1=/HK1
ROOT-
bootstrap.RegisteredSubDomainForkProcess_1=topic://FORKDAEMON/ATGLOBAL-
HK1/:65535
ROOT-bootstrap.RegisteredSubDomain_2=/HK2
ROOT-
bootstrap.RegisteredSubDomainForkProcess_2=topic://FORKDAEMON/ATGLOBAL-
HK2/:65535

# Location of ForkProcess Daemons for spawning Managers
ROOT-bootstrap.NumberOfForkDaemons=0

# Location of Messaging Node
ROOT-bootstrap.NumberOfMessagingNodeDaemons=1
ROOT-bootstrap.MessagingNode_1=127.0.0.1

# -----
# Config Entries for HK1 - Bootstrap Node
# -----

# The domain of the bootstrap program
ROOT_HK1-bootstrap.Level=/HK1

# Number of registered subDomains
ROOT_HK1-bootstrap.NumOfRegisteredSubDomains=0

# Domain URI of subDomains and their locations

# Registry Locator
```

A unique string for ForkDaemon. It is used as NB topic

Locator of ROOT's ForkDaemon.

Altogether 2 sub-domains

Locators of sub-domains' ForkDaemons

Name of the domain. Notice how the name appears at the start of the line

```
ROOT_HK1-bootstrap.RegistryForkDaemon=topic://FORKDAEMON/ATGLOBAL-  
HK1/:65535
```

```
ROOT_HK1-bootstrap.RegistryPersistentStore=wscontext:HK1
```

If this line is present,
WS-Context is used as
persistent storage.
Otherwise everything is
just saved in memory

```
# Location of Messaging Node
```

```
ROOT_HK1-bootstrap.NumberOfMessagingNodeDaemons=1
```

```
ROOT_HK1-bootstrap.MessagingNode_1=127.0.0.1
```

```
# Location of ForkProcess Daemons for spawning Managers
```

```
ROOT_HK1-bootstrap.NumberOfForkDaemons=1
```

```
ROOT_HK1-bootstrap.ForkDaemon_1=topic://FORKDAEMON/ATGLOBAL-HK1/:65535
```

```
# -----
```

```
# Config Entries for Service Adapter
```

```
# -----
```

```
ServiceAdapter.NumOfMessagingNodes=1
```

```
ServiceAdapter.MessagingNode_1=127.0.0.1
```

```
ServiceAdapter.Level=HK1
```

Connect to local
Messaging Node

Sensors will be
deployed to this
domain

```
# -----
```

```
# Config Entries for User Console
```

```
# -----
```

```
user.MessagingNode=127.0.0.1
```

```
user.MessagingNodePort=25050
```

```
user.MessagingNodeTransport=niotcp
```

```
user.RegistryMonitorInterval=30000
```

```
# -----
```

```
# Config Entries for BootStrapService UI
```

```
# -----
```

```
BootStrapServiceUI.MessagingNode=127.0.0.1
```

```
BootStrapServiceUI.MessagingNodePort=25050
```

```
BootStrapServiceUI.MessagingNodeTransport=niotcp
```

Configuration file of Domain HK2

```
# -----
```

```
# Config Entries for Fork Daemon
```

```
# -----
```

```
## This string should be unique for different networks
```

```
## It is used to uniquely identify a Fork Daemon
```

```
ForkDaemon.UniqueString=ATGLOBAL-HK2
```

A unique string for
ForkDaemon. It is
used as NB topic

```
# -----
```

```
# Config Entries for HK2 - Bootstrap Node
```

```
# -----
```

```
# The domain of the bootstrap program
```

```
ROOT_HK2-bootstrap.Level=HK2
```

Name of the
domain

```
# Number of registered subDomains
```

```
ROOT_HK2-bootstrap.NumOfRegisteredSubDomains=0
```

```
# Domain URI of subDomains and their locations
```

```
# Registry Locator
```

```

ROOT_HK2-bootstrap.RegistryForkDaemon=topic://FORKDAEMON/ATGLOBAL-
HK2/:65535
ROOT_HK2-bootstrap.RegistryPersistentStore=wscontext:HK2

# Locaton of Messaging Node
ROOT_HK2-bootstrap.NumberOfMessagingNodeDaemons=1
ROOT_HK2-bootstrap.MessagingNode_1=127.0.0.1

# Locaton of ForkProcess Daemons for spawning Managers
ROOT_HK2-bootstrap.NumberOfForkDaemons=1
ROOT_HK2-bootstrap.ForkDaemon_1=topic://FORKDAEMON/ATGLOBAL-HK2/:65535

# -----
# Config Entries for Service Adapter
# -----
ServiceAdapter.NumOfMessagingNodes=1
ServiceAdapter.MessagingNode_1=127.0.0.1
ServiceAdapter.Level=/HK2

# -----
# Config Entries for User Console
# -----
user.MessagingNode=127.0.0.1
user.MessagingNodePort=25050
user.MessagingNodeTransport=niotcp
user.RegistryMonitorInterval=30000

# -----
# Config Entries for BootStrapService UI
# -----
BootStrapServiceUI.MessagingNode=127.0.0.1
BootStrapServiceUI.MessagingNodePort=25050
BootStrapServiceUI.MessagingNodeTransport=niotcp

```

D.5.2 defaultMessagingNode.conf

The full path of this file is *GridBuilder/conf/defaultMessagingNode.conf*. This configuration file is used to define various properties of the Messaging Node. The Messaging Node determines which NB network the domain connects to. You **SHOULD** connect the domain to a messaging node which connects directly or indirectly to the Sensor Grid server.

Here is a sample file.

```

# -----
# Prioritized Protocols
# -----

PRIORITIZED_PROTOCOL_LIST.prioritizedProtocolList=niotcp,tcp,udp,http,https,ssl

# -----
# Default Messaging Node properties
# -----

```

```

DEFAULT_MESSAGING_NODE.NIOTCPBrokerPort=25050
DEFAULT_MESSAGING_NODE.TCPBrokerPort=25060
DEFAULT_MESSAGING_NODE.UDPBrokerPort=25070
DEFAULT_MESSAGING_NODE.HTTPBrokerPort=0
DEFAULT_MESSAGING_NODE.HTTPSBrokerPort=0
DEFAULT_MESSAGING_NODE.SSLBrokerPort=0
DEFAULT_MESSAGING_NODE.PTCPBrokerPort=0
DEFAULT_MESSAGING_NODE.MulticastGroupPort=0
DEFAULT_MESSAGING_NODE.MulticastGroupHost=224.224.224.224
DEFAULT_MESSAGING_NODE.PoolTCPBrokerPort=0
DEFAULT_MESSAGING_NODE.PTCPStreamNumber=5
DEFAULT_MESSAGING_NODE.AssignedAddress=false
DEFAULT_MESSAGING_NODE.NodeAddress=1,1,1,1
DEFAULT_MESSAGING_NODE.VirtualBrokerNetwork=network-CGL-1
DEFAULT_MESSAGING_NODE.SupportRTP=no
DEFAULT_MESSAGING_NODE.BDNList=
DEFAULT_MESSAGING_NODE.ConcurrentConnectionLimit=3000
DEFAULT_MESSAGING_NODE.Discriminator=156.56.*
DEFAULT_MESSAGING_NODE.AboutThisBroker=Default Messaging Node
DEFAULT_MESSAGING_NODE.MAXBrokerDiscoRequests=1000
DEFAULT_MESSAGING_NODE.DiscoveryResponsePolicy=cgl.narada.discovery.bro
ker.DefaultBrokerDiscoveryRequestResponsePolicy
DEFAULT_MESSAGING_NODE.BrokerKeyStore=keystore/NBSecurityTest.keys

# These are required only if AssignedAddress is false
DEFAULT_MESSAGING_NODE.ConnectAddress=202.94.237.242
DEFAULT_MESSAGING_NODE.ConnectTransport=niotcp
DEFAULT_MESSAGING_NODE.ConnectPort=25050

```

Messaging Node
connects to this
address

D.5.3 setEnv.bat

The full path of this file is *GridBuilder/bin/setEnv.bat*. This configuration file setups environment variables that will be used by Grid Builder. You would only have to modify the path of package NaradaBroker-3.2.0.

Here is a sample file:

```

@echo off

REM NOTES:
REM -----
REM March 24, 2005
REM Modified to put saaj.jar before other AXIS jars. because without
REM this, the system gives a java.lang.IncompatibleClassChangeError
REM Ref: http://buzz.bowstreet.com/snitz/topic.asp?TOPIC_ID=500

REM Sets the environment variables. This must be called from all
REM run*.bat
REM files to set the proper environment
REM -----

REM To enable Asynchronous WsContext service set this to -async
REM For Sync version, set to blank
REM -----

```

```
set HPSEARCH_HOME=..\
set NB_HOME=..\..\NaradaBrokering-3.2.0
```

Path of NB 3.2 package. Should be included in the GB package

```
REM -----
REM Set the classpath
REM Please make path changes IIF and AS required

REM NOTE: For the "FOR ..." command to work, The cmd.exe must
REM       be started using the /vk parameter, else use the cpappend below
REM FOR %%j IN (%HPSEARCH_HOME%\lib\*.jar) do set cp=!cp!;%%j
REM -----

set cp=%HPSEARCH_HOME%\lib\saa.jar

FOR %%i IN ("%HPSEARCH_HOME%\lib\*.jar") DO CALL cpappend.bat %%i
FOR %%i IN ("%NB_HOME%\lib\*.jar") DO CALL cpappend.bat %%i

set path=%path%;%NB_HOME%\dll
```

D.6 Step by Step Domain Deployment Guide

In this section you are going to walkthrough the deployment steps for setting up 3 domains namely ISAAC, XPS and 5150, with ISAAC as the ROOT Domain. Figure 9-8 shows the overview of domains we are going to setup.

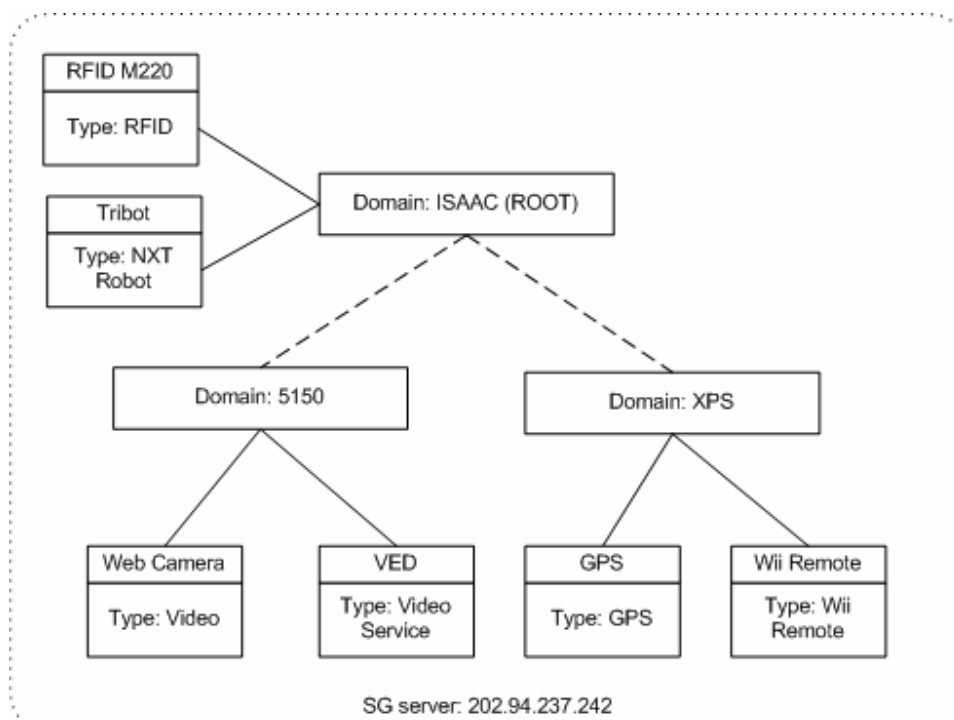


Figure 9-8 Step by step ensor deployment overview

Please follow the steps below to deploy the domains.

Step 1 – Set Up Domains

Before you start, make sure that a Sensor Grid server is up and running. For instructions on setting up a Sensor Grid server please refer to Appendix C -

User Guide for System Administrator.

Suppose a Sensor Grid Server has been set up with IP address 202.94.237.242. Prepare 3 PCs each with its own unique IP address and fulfills the system and network requirements. The Grid Builder package should also be installed on each of the PCs to a path without any space characters (e.g. c:\GBPackage). From now on we will refer to this directory as **<installation directory>**.

Step 2 – Configure defaultMessagingNode.conf

For all domains, use the following configuration in
<installation directory>\GridBuilder\conf\defaultMessagingNode.conf

```
# -----
# Prioritized Protocols
# -----

PRIORITIZED_PROTOCOL_LIST.prioritizedProtocolList=niotcp,tcp,udp,http,https,ssl

# -----
# Default Messaging Node properties
# -----

DEFAULT_MESSAGING_NODE.NIOTCPBrokerPort=25050
DEFAULT_MESSAGING_NODE.TCPBrokerPort=25060
DEFAULT_MESSAGING_NODE.UDPBrokerPort=25070
DEFAULT_MESSAGING_NODE.HTTPBrokerPort=0
DEFAULT_MESSAGING_NODE.HTTPSBrokerPort=0
DEFAULT_MESSAGING_NODE.SSLBrokerPort=0
DEFAULT_MESSAGING_NODE.PTCPBrokerPort=0
DEFAULT_MESSAGING_NODE.MulticastGroupPort=0
DEFAULT_MESSAGING_NODE.MulticastGroupHost=224.224.224.224
DEFAULT_MESSAGING_NODE.PoolTCPBrokerPort=0
DEFAULT_MESSAGING_NODE.PTCPStreamNumber=5
DEFAULT_MESSAGING_NODE.AssignedAddress=false
DEFAULT_MESSAGING_NODE.NodeAddress=1,1,1,1
DEFAULT_MESSAGING_NODE.VirtualBrokerNetwork=network-CGL-1
DEFAULT_MESSAGING_NODE.SupportRTP=no
DEFAULT_MESSAGING_NODE.BDNList=
DEFAULT_MESSAGING_NODE.ConcurrentConnectionLimit=3000
DEFAULT_MESSAGING_NODE.Discriminator=156.56.*
DEFAULT_MESSAGING_NODE.AboutThisBroker=Default Messaging Node
DEFAULT_MESSAGING_NODE.MAXBrokerDiscoRequests=1000
DEFAULT_MESSAGING_NODE.DiscoveryResponsePolicy=cgl.narada.discovery.broker.DefaultBrokerDiscoveryRequestResponsePolicy
DEFAULT_MESSAGING_NODE.BrokerKeyStore=keystore/NBSecurityTest.keys

# These are required only if AssignedAddress is false
DEFAULT_MESSAGING_NODE.ConnectAddress=202.94.237.242
DEFAULT_MESSAGING_NODE.ConnectTransport=niotcp
DEFAULT_MESSAGING_NODE.ConnectPort=25050
```

Enter the address of
SG server here

Step 3 – Configure setEnv.bat

For all domains, make sure that you have setup the correct path for NaradaBrokering-3.2.0 package in <installation directory>\GridBuilder\bin\setEnv.bat

```
@echo off

REM NOTES:
REM -----
REM March 24, 2005
REM Modified to put saaj.jar before other AXiS jars. because without
REM this, the system gives a java.lang.IncompatibleClassChangeError
REM Ref: http://buzz.bowstreet.com/snitz/topic.asp?TOPIC_ID=500

REM Sets the environment variables. This must be called from all
run*.bat
REM files to set the proper environment
REM -----

REM To enable Asynchronous WsContext service set this to -async
REM For Sync version, set to blank
REM -----

set HPSEARCH_HOME=..\
set NB_HOME=..\..\NaradaBrokering-3.2.0

REM -----
REM Set the classpath
REM Please make path changes 1IF and AS required

REM NOTE: For the "FOR ..." command to work, The cmd.exe must
REM       be started using the /vk parameter, else use the cpappend below
REM FOR %%j IN (%HPSEARCH_HOME%\lib\*.jar) do set cp=!cp!;%%j
REM -----

set cp=%HPSEARCH_HOME%\lib\saa.jar

FOR %%i IN ("%HPSEARCH_HOME%\lib\*.jar") DO CALL cpappend.bat %%i
FOR %%i IN ("%NB_HOME%\lib\*.jar") DO CALL cpappend.bat %%i

set path=%path%;%NB_HOME%\dll
```

Enter the correct
path of NB 3.2 here

Step 4 – Configure mgmtSystem.conf

For each domain, the corresponding configuration in <installation directory>\GridBuilder\conf\mgmtSystem.conf should be different. The corresponding configuration files of the 3 domains are shown below:

Domain ISAAC:

```
# -----
# Config Entries for Fork Daemon
# -----
## This string should be unique for different networks
```

```

## It is used to uniquely identify a Fork Daemon
ForkDaemon.UniqueString=ATGLOBAL-ISAAC

# -----
# Config Entries for Root Bootstrap Node
# -----

# The domain of the bootstrap program
ROOT-bootstrap.Level=/
ROOT-bootstrap.ForkProcessLocator=topic://FORKDAEMON/ATGLOBAL-
ISAAC/:65535

# Number of registered subDomains
ROOT-bootstrap.NumOfRegisteredSubDomains=3

# Domain URI of subDomains and their locations
ROOT-bootstrap.RegisteredSubDomain_1=/ISAAC
ROOT-
bootstrap.RegisteredSubDomainForkProcess_1=topic://FORKDAEMON/ATGLOBAL-
ISAAC/:65535
ROOT-bootstrap.RegisteredSubDomain_2=/XPS
ROOT-
bootstrap.RegisteredSubDomainForkProcess_2=topic://FORKDAEMON/ATGLOBAL-
XPS/:65535
ROOT-bootstrap.RegisteredSubDomain_3=/5150
ROOT-
bootstrap.RegisteredSubDomainForkProcess_3=topic://FORKDAEMON/ATGLOBAL-
5150/:65535
ROOT-bootstrap.RegisteredSubDomain_4=/INDIANA
ROOT-
bootstrap.RegisteredSubDomainForkProcess_4=topic://FORKDAEMON/ATGLOBAL-
INDIANA/:65535

# Locaton of ForkProcess Daemons for spawning Managers
ROOT-bootstrap.NumberOfForkDaemons=0

# Locaton of Messaging Node
ROOT-bootstrap.NumberOfMessagingNodeDaemons=1
ROOT-bootstrap.MessagingNode_1=127.0.0.1

# -----
# Config Entries for ISAAC - Bootstrap Node
# -----

# The domain of the bootstrap program
ROOT_ISAAC-bootstrap.Level=/ISAAC

# Number of registered subDomains
ROOT_ISAAC-bootstrap.NumOfRegisteredSubDomains=0

# Domain URI of subDomains and their locations

# Registry Locator
ROOT_ISAAC-bootstrap.RegistryForkDaemon=topic://FORKDAEMON/ATGLOBAL-
ISAAC/:65535
ROOT_ISAAC-bootstrap.RegistryPersistentStore=wscontext:ISAAC

```

```

# Locaton of Messaging Node
ROOT_ISAAC-bootstrap.NumberOfMessagingNodeDaemons=1
ROOT_ISAAC-bootstrap.MessagingNode_1=127.0.0.1

# Locaton of ForkProcess Daemons for spawning Managers
ROOT_ISAAC-bootstrap.NumberOfForkDaemons=1
ROOT_ISAAC-bootstrap.ForkDaemon_1=topic://FORKDAEMON/ATGLOBAL-
ISAAC/:65535

# -----
# Config Entries for Service Adapter
# -----
ServiceAdapter.NumOfMessagingNodes=1
ServiceAdapter.MessagingNode_1=127.0.0.1
ServiceAdapter.Level=/ISAAC

# -----
# Config Entries for User Console
# -----
user.MessagingNode=127.0.0.1
user.MessagingNodePort=25050
user.MessagingNodeTransport=niotcp
user.RegistryMonitorInterval=30000

# -----
# Config Entries for BootStrapService UI
# -----
BootStrapServiceUI.MessagingNode=127.0.0.1
BootStrapServiceUI.MessagingNodePort=25050
BootStrapServiceUI.MessagingNodeTransport=niotcp

```

Domain XPS:

```

# -----
# Config Entries for Fork Daemon
# -----
## This string should be unique for different networks
## It is used to uniquely identify a Fork Daemon
ForkDaemon.UniqueString=ATGLOBAL-XPS

# -----
# Config Entries for XPS - Bootstrap Node
# -----

# The domain of the bootstrap program
ROOT_XPS-bootstrap.Level=/XPS

# Number of registered subDomains
ROOT_XPS-bootstrap.NumOfRegisteredSubDomains=0

# Domain URI of subDomains and their locations

# Registry Locator
ROOT_XPS-bootstrap.RegistryForkDaemon=topic://FORKDAEMON/ATGLOBAL-
XPS/:65535
ROOT_XPS-bootstrap.RegistryPersistentStore=wscontext:XPS

```

```

# Locaton of Messaging Node
ROOT_XPS-bootstrap.NumberOfMessagingNodeDaemons=1
ROOT_XPS-bootstrap.MessagingNode_1=127.0.0.1

# Locaton of ForkProcess Daemons for spawning Managers
ROOT_XPS-bootstrap.NumberOfForkDaemons=1
ROOT_XPS-bootstrap.ForkDaemon_1=topic://FORKDAEMON/ATGLOBAL-XPS/:65535

# -----
# Config Entries for Service Adapter
# -----
ServiceAdapter.NumOfMessagingNodes=1
ServiceAdapter.MessagingNode_1=127.0.0.1
ServiceAdapter.Level=/XPS

# -----
# Config Entries for User Console
# -----
user.MessagingNode=127.0.0.1
user.MessagingNodePort=25050
user.MessagingNodeTransport=niotcp
user.RegistryMonitorInterval=30000

# -----
# Config Entries for BootStrapService UI
# -----
BootStrapServiceUI.MessagingNode=127.0.0.1
BootStrapServiceUI.MessagingNodePort=25050
BootStrapServiceUI.MessagingNodeTransport=niotcp

```

Domain 5150:

```

# -----
# Config Entries for Fork Daemon
# -----
## This string should be unique for different networks
## It is used to uniquely identify a Fork Daemon
ForkDaemon.UniqueString=ATGLOBAL-5150

# -----
# Config Entries for 5150 - Bootstrap Node
# -----

# The domain of the bootstrap program
ROOT_5150-bootstrap.Level=/5150

# Number of registered subDomains
ROOT_5150-bootstrap.NumOfRegisteredSubDomains=0

# Domain URI of subDomains and their locations

# Registry Locator
ROOT_5150-bootstrap.RegistryForkDaemon=topic://FORKDAEMON/ATGLOBAL-
5150/:65535
ROOT_5150-bootstrap.RegistryPersistentStore=wscontext:5150

```

```

# Locaton of Messaging Node
ROOT_5150-bootstrap.NumberOfMessagingNodeDaemons=1
ROOT_5150-bootstrap.MessagingNode_1=127.0.0.1

# Locaton of ForkProcess Daemons for spawning Managers
ROOT_5150-bootstrap.NumberOfForkDaemons=1
ROOT_5150-bootstrap.ForkDaemon_1=topic://FORKDAEMON/ATGLOBAL-
5150/:65535

# -----
# Config Entries for Service Adapter
# -----
ServiceAdapter.NumOfMessagingNodes=1
ServiceAdapter.MessagingNode_1=127.0.0.1
ServiceAdapter.Level=/5150

# -----
# Config Entries for User Console
# -----
user.MessagingNode=127.0.0.1
user.MessagingNodePort=25050
user.MessagingNodeTransport=niotcp
user.RegistryMonitorInterval=30000

# -----
# Config Entries for BootStrapService UI
# -----
BootStrapServiceUI.MessagingNode=127.0.0.1
BootStrapServiceUI.MessagingNodePort=25050
BootStrapServiceUI.MessagingNodeTransport=niotcp

```

Step 5 – Configuring WS-Context

To use WS-Context, the following configuration files have to be modified.

wscontext.properties

This file is located at *<installation directory>\GridBuilder\conf\wscontext.properties*. Please configure it as followed. You only have to pay attention to text highlighted in red.

```

#####
#
#
# FTHPIS - Property file used to set parameters for UDDI-Extended
# Information Service
# Web Site: http://grids.ucs.indiana.edu/~maktas/fthpis/index.html
#
#####
##
#
# JDBC Connection parameters
#
#####
##

```

```

cgl.fthpis.useConnectionPool = true
cgl.fthpis.jdbcDriver      = org.gjt.mm.mysql.Driver
cgl.fthpis.wscontext.jdbcURL =
jdbc:mysql://127.0.0.1:3306/cgl_wscontext
cgl.fthpis.uddi.jdbcURL    = jdbc:mysql://127.0.0.1:3306/cgl_uddi
cgl.fthpis.jdbcMaxActive = 10
cgl.fthpis.jdbcMaxIdle   = 5

#####
##
#
# Userid/passwords should not generally be stored in clear text
#
#####
##

cgl.fthpis.jdbcUser = uddi_user
cgl.fthpis.jdbcPassword = changeIt

#####
##
#
# DataStore Modules
#
#####
##

# DataStore module currently to use
juddi.dataStore = org.apache.juddi.datastore.jdbc.JDBCDataStore
# ExtendedUDDI DataStore module currently to use
juddi.extendeduddiDataStore = cgl.fthpis.datastore.jdbc.JDBCDataStore
# WSontext DataStore module currently to use
juddi.wscontextDataStore =
cgl.fthpis.datastore.jdbc.WSContextJDBCDataStore

#####
##
#
# FTHPIS SYSTEM paramaters
#
#####
##

fthpis.type = 1
#1-centralized , 2-decentralized

#mappingFile.path = C:/MyApps/HybridService/mappingfiles
mappingFile.path = C:/GBPackage/wscontext_exe/apache-tomcat-
5.5.26/webapps/axis2/WEB-INF/classes/HybridService/mappingfiles

default.infoservice = UDDI
#default.infoservice = WS-CONTEXT

#####
##
#
# The WSDL address for the inquiry and publishing API of the target

```

Change this to your
own installation
directory

```

# UDDI-Extended Information Service
#
#####
##

UDDI_WSContext_WSDL =
http://localhost:8080/axis2/services/HYBRID_SERVICE
#UDDI_WSContext_WSDL =
http://gf12.ucs.indiana.edu:4780/axis2/services/HYBRID_SERVICE
#UDDI_WSContext_WSDL =
http://gf6.ucs.indiana.edu:4347/axis2/services/HYBRID_SERVICE

#####
##
#
# Debug log enabled or not.
# OFF/INFO
#####
##

logLevel=INFO

#####
##
#
# BENCHMARK
#####
##

#eger publication test ediyorsak bu true olucak. inquiry ise false
olucak
publication_benchmark=true
#eger inquiry test ediyorsak bu true olucak. publication ise false
olucak
inquiry_benchmark=false

#####
##
#
# PUB-SUB System paramaters
#
#####
##

# CACHE INFO - 20 MB = 1024 x 1024 x 20 = 20,971,520
highwatermark = 20971520

fthpis.timeout = 10000
fthpis.replicaset = 1

#####
##
# NB Parameters. Please replace following NB parameters to point to
your
# Narada Broker

```

```
#####
##

#FTHPIShostname = gf2.ucs.indiana.edu
#FTHPISID = 2
#hostname = gf6.ucs.indiana.edu
#portnum = 4648
#protocol = niotcp
#NB_HOME=/home/maktas/nb/NaradaBrokering-1.1.6

FTHPIShostname = localhost
FTHPISID = 1
hostname = localhost
portnum = 3035
protocol = niotcp
NB_HOME=C:/GBPackage/wscontext_exe/NaradaBrokering-3.2.0

#####
###
# NB Service Configuration Parameters
#####
###

#This specifies the location of the Fragmentation Directory needed by
FragmentationDirectory=C:/TempFiles/tmpFiles/fragment

#This specifies the location of the coalescing directory
CoalescingDirectory=C:/TempFiles/tmpFiles/coalesce

#This specifies the location of the Security keystore
SecurityKeyStore=C:/SecurityStores/keystore

#This specifies the location of the Security truststore
SecurityTrustStore=C:/SecurityStores/truststore

#This specifies the cryptography provider within the system
SecurityProvider=CryptixCrypto

#Specifies the location of the stratum-1 time servers used by entities.
#time-a.nist.gov,time-b.nist.gov,time-a.timefreq.bldrdoc.gov,time-
b.timefreq.bldrdoc.gov,
#time-c.timefreq.bldrdoc.gov,time.nist.gov,time-
nw.nist.gov,utcnist.colorado.edu
# ,131.107.1.10,128.138.140.44
#NTP_Servers =
129.6.15.28,129.6.15.29,132.163.4.101,132.163.4.102,132.163.4.103,192.4
3.244.18
NTP_Servers =
```

Change this to your
own installation
directory


```

## This is the time interval (milliseconds) between successive runs of
the NTP synchronization with an NTP time server,
## The default value is 30 seconds.
#NTP_Interval=2000
NTP_Interval=30000

NTP_Debug=OFF

#These pertain to Reliable Delivery Service Implementations
(db=Database, file=FileStorage)
Storage_Type=db

Database_JDBC_Driver=org.gjt.mm.mysql.Driver
Database_ConnectionProvider=jdbc:mysql
Database_ConnectionHost=localhost
Database_ConnectionPort=3306
Database_ConnectionDatabase=NbPersistence

FileStorage_BaseDirectory=C:/NBStorage/filebased/persistent

TOB_MaximumTotalBufferSize=2500000

TOB_MaximumNumberOfBufferEntries=10000

#In milliseconds#
TOB_MaximumBufferEntryDuration=50000
TOB_BufferReleaseFactor=0.8

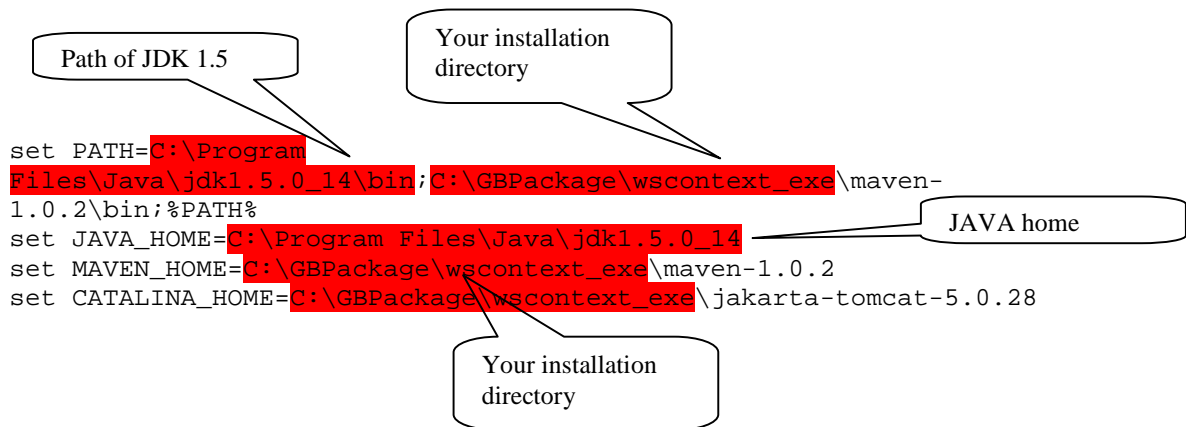
# Comma seperated list of publicly known Broker Discovery Services
#
BDNDiscoveryList=http://www.idonotexist.com,http://trex.ucs.indiana.edu
:8080/BDN/servlet/Discover,http://www.grid servicelocator.org/

MulticastGroupHost=224.224.224.224
MulticastGroupPort=0

```

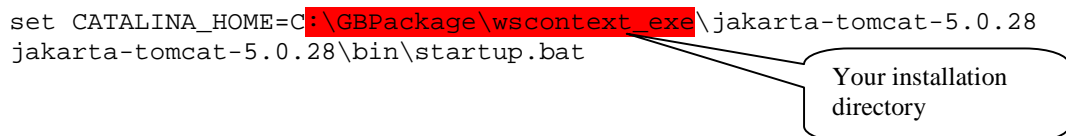
env.bat

This file is located at *<installation directory>\wscontext_exe\env.bat*. Please configure it as followed. You only have to pay attention to text highlighted in red.



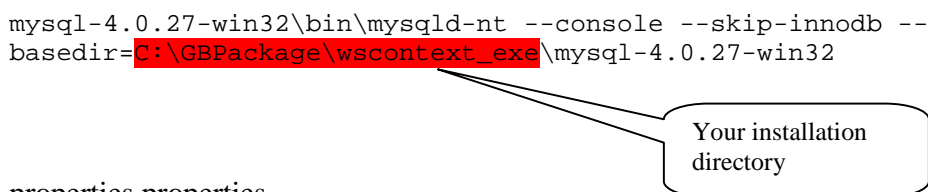
startjakartatomcat.bat

This file is located at *<installation directory>\wscontext_exe\startjakartatomcat.bat*. Please configure it as followed. You only have to pay attention to text highlighted in red.



startmysql4.0.bat

This file is located at *<installation directory>\wscontext_exe\startmysql4.0.bat*. Please configure it as followed. You only have to pay attention to text highlighted in red.



properties.properties

This file is located at *<installation directory>\wscontext_exe\jakarta-tomcat-5.0.28\webapps\axis2\WEB-INF\classes\properties.properties*. Please configure it as followed. You only have to pay attention to text highlighted in red.

```
#####
#
#
# FTHPIS - Property file used to set parameters for UDDI-Extended
# Information Service
# Web Site: http://grids.ucs.indiana.edu/~maktas/fthpis/index.html
#
#####
##
```

```

#
# JDBC Connection parameters
#
#####
##

cgl.fthpis.useConnectionPool = true
cgl.fthpis.jdbcDriver      = org.gjt.mm.mysql.Driver
#cgl.fthpis.wscontext.jdbcURL  =
jdbc:mysql://gf7.ucs.indiana.edu:3306/cgl_wscontext
#cgl.fthpis.uddi.jdbcURL      =
jdbc:mysql://gf7.ucs.indiana.edu:3306/cgl_uddi
cgl.fthpis.wscontext.jdbcURL  =
jdbc:mysql://127.0.0.1:3306/cgl_wscontext_deneme
cgl.fthpis.uddi.jdbcURL      = jdbc:mysql://127.0.0.1:3306/cgl_uddi
cgl.fthpis.jdbcMaxActive = 10
cgl.fthpis.jdbcMaxIdle   = 5

#####
##
#
# Userid/passwords should not generally be stored in clear text
#
#####
##

cgl.fthpis.jdbcUser = uddi_user
cgl.fthpis.jdbcPassword = changeIt

#####
##
#
# DataStore Modules
#
#####
##

# DataStore module currently to use
juddi.dataStore = org.apache.juddi.datastore.jdbc.JDBCDataStore
# ExtendedUDDI DataStore module currently to use
juddi.extendeduddiDataStore = cgl.fthpis.datastore.jdbc.JDBCDataStore
# WSontext DataStore module currently to use
juddi.wscontextDataStore =
cgl.fthpis.datastore.jdbc.WSContextJDBCDataStore

#####
##
#
# FTHPIS SYSTEM paramaters
#
#####
##

fthpis.type = 1
#1-centralized , 2-decentralized

#mappingFile.path = C:/MyApps/HybridService/mappingfiles

```

Your installation
directory

```

mappingFile.path = C:/GBPackage/wscontext_exe/jakarta-tomcat-
5.0.28/webapps/axis2/WEB-INF/classes/HybridService/mappingfiles

default.infoservice = UDDI
#default.infoservice = WS-CONTEXT

#####
##
#
# The WSDL address for the inquiry and publishing API of the target
# UDDI-Extended Information Service
#
#####
##

#UDDI_Extended_WSDL =
http://localhost:8080/uddi_wscontext/services/UDDI_Extended
UDDI_WSContext_WSDL =
http://localhost:8080/axis2/services/UDDI_WSContextService
#UDDI_WSContext_WSDL =
http://gf6.ucs.indiana.edu:4347/axis2/services/UDDI_WSContextService

#UDDI_Extended_WSDL =
http://gf6.ucs.indiana.edu:4347/uddi_wscontext/services/UDDI_Extended
#UDDI_WSContext_WSDL =
http://gf6.ucs.indiana.edu:4347/uddi_wscontext/services/UDDI_WSContext

#UDDI_Extended_WSDL =
http://gf8.ucs.indiana.edu:4647/uddi_wscontext/services/UDDI_Extended
#UDDI_WSContext_WSDL =
http://gf8.ucs.indiana.edu:4647/uddi_wscontext/services/UDDI_WSContext

#UDDI_Extended_WSDL =
http://gf8.ucs.indiana.edu:4947/uddi_wscontext/services/UDDI_Extended
#UDDI_WSContext_WSDL =
http://gf8.ucs.indiana.edu:4947/uddi_wscontext/services/UDDI_WSContext

#####
##
#
# Debug log enabled or not.
# OFF/INFO
#####
##

logLevel=INFO

#####
##
#
# BENCHMARK
#####
##

#eger publication test ediyorsak bu true olucak. inquiry ise false
olucak
publication_benchmark=true

```

```

##eger inquiry test ediyorsak bu true olucak. publication ise false
olucak
inquiry_benchmark=false

#####
##
#
# PUB-SUB System paramaters
#
#####
##

# CACHE INFO - 20 MB = 1024 x 1024 x 20 = 20,971,520
highwatermark = 20971520

fthpis.timeout = 10000
fthpis.replicaset = 1

#####
##
# NB Parameters. Please replace following NB parameters to point to
your
# Narada Broker
#####
##

#FTHPIShostname = gf2.ucs.indiana.edu
#FTHPISID = 2
#hostname = gf6.ucs.indiana.edu
#portnum = 4648
#protocol = niotcp
#NB_HOME=/home/maktas/nb/NaradaBrokering-1.1.6

FTHPIShostname = localhost
FTHPISID = 1
hostname = localhost
portnum = 4648
protocol = niotcp
NB_HOME=C:/GBPackage/wscontext_exe/NaradaBrokering-3.2.0

#####
###
# NB Service Configuration Parameters
#####
###

#This specifies the location of the Fragmentation Directory needed by
FragmentationDirectory=C:/TempFiles/tmpFiles/fragment

#This specifies the location of the coalescing directory
CoalescingDirectory=C:/TempFiles/tmpFiles/coalesce

#This specifies the location of the Security keystore

```

Your installation
directory

```

SecurityKeyStore=C:/SecurityStores/keystore

#This specifies the location of the Security truststore
SecurityTrustStore=C:/SecurityStores/truststore

#This specifies the cryptography provider within the system
SecurityProvider=CryptixCrypto

#Specifies the location of the stratum-1 time servers used by entities.
#time-a.nist.gov,time-b.nist.gov,time-a.timefreq.bldrdoc.gov,time-
b.timefreq.bldrdoc.gov,
#time-c.timefreq.bldrdoc.gov,time.nist.gov,time-
nw.nist.gov,utcnist.colorado.edu
# ,131.107.1.10,128.138.140.44
#NTP_Servers =
129.6.15.28,129.6.15.29,132.163.4.101,132.163.4.102,132.163.4.103,192.4
3.244.18
NTP_Servers =

## This is the time interval (milliseconds) between successive runs of
the NTP synchronization with an NTP time server,
## The default value is 30 seconds.
#NTP_Interval=2000
NTP_Interval=30000

NTP_Debug=OFF

#These pertain to Reliable Delivery Service Implementations
(db=Database, file=FileStorage)
Storage_Type=db

Database_JDBC_Driver=org.gjt.mm.mysql.Driver
Database_ConnectionProvider=jdbc:mysql
Database_ConnectionHost=localhost
Database_ConnectionPort=3306
Database_ConnectionDatabase=NbPersistence

FileStorage_BaseDirectory=C:/NBStorage/filebased/persistent

TOB_MaximumTotalBufferSize=2500000

TOB_MaximumNumberOfBufferEntries=10000

#In milliseconds#
TOB_MaximumBufferEntryDuration=50000
TOB_BufferReleaseFactor=0.8

# Comma seperated list of publicly known Broker Discovery Services

```

```
#
BDNDDiscoveryList=http://www.idonotexist.com,http://trex.ucs.indiana.edu
:8080/BDN/servlet/Discover,http://www.gridservicelocator.org/
```

```
MulticastGroupHost=224.224.224.224
MulticastGroupPort=0
```

setpath.bat

This file is located at <installation directory>\wscontext_exe\
uddi_wscontext_services_v5\setpath.bat. Please configure it as followed. You only have
to pay attention to text highlighted in red.

```
::set AXIS2_HOME=D:\wscontext\axis2-1.3
::set AXIS2_HOME=D:\wscontext\axis2-1.2
::set AXIS2_HOME=D:\wscontext\axis2-1.1.1
::set AXIS2_HOME=D:\wscontext\axis2-1.1
::set AXIS2_HOME=D:\wscontext\axis2-std-1.0-bin
set
AXIS2_HOME=C:\GBPackage\wscontext_exe\uddi_wscontext_services_v5\axis_1
ib\axis2_SNAPSHOT
```

Your installation
directory

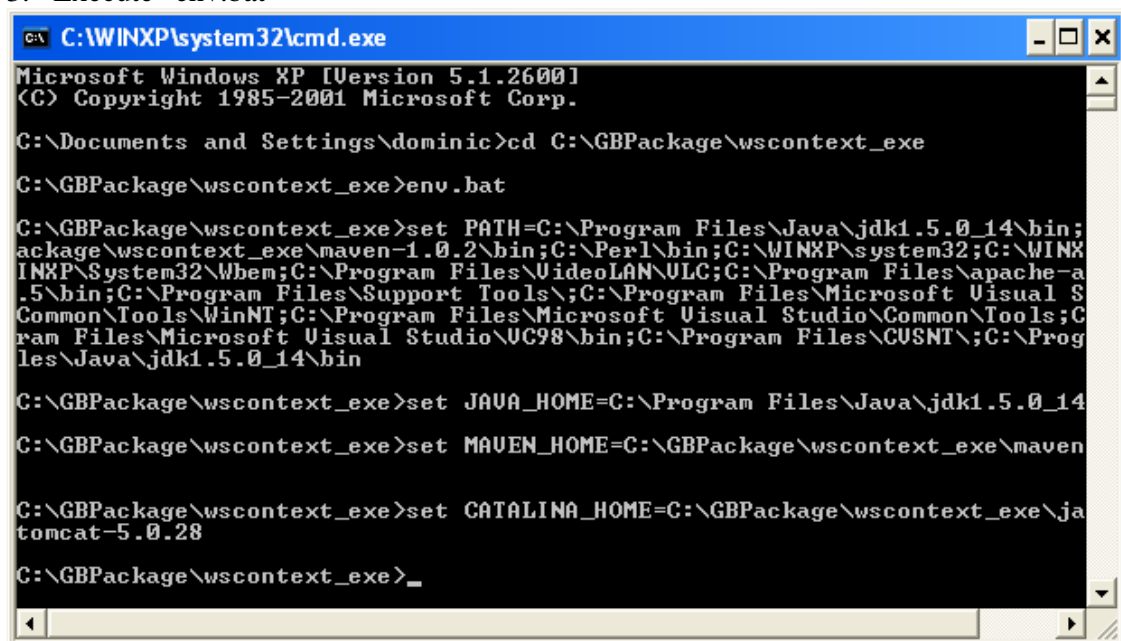
Step 6 – Starting WS-Context

We are now ready to start the servers for WS-Context. Please follow the instructions
below to start and verify all components are setup correctly.

Starting MySQL

Please follow these steps to start MySQL:

1. Open “cmd” by clicking Windows’ “Start” -> Run -> type “cmd” -> OK
2. Go to <installation directory>\wscontext_exe
3. Execute “env.bat”



```
C:\WINXP\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

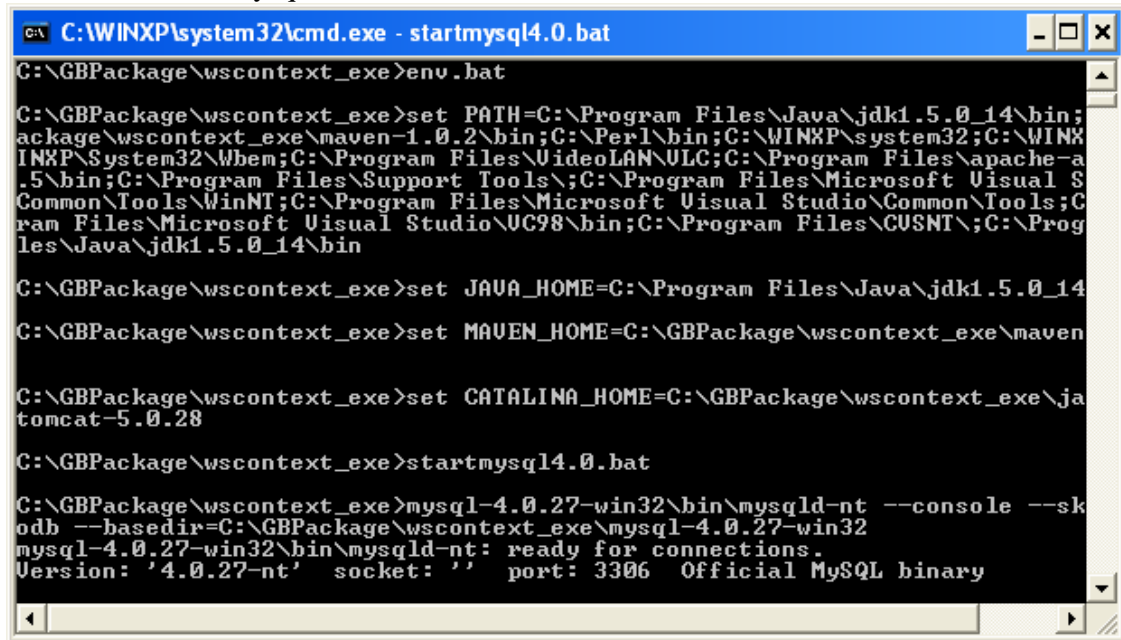
C:\Documents and Settings\dominic>cd C:\GBPackage\wscontext_exe
C:\GBPackage\wscontext_exe>env.bat

C:\GBPackage\wscontext_exe>set PATH=C:\Program Files\Java\jdk1.5.0_14\bin;
ackage\wscontext_exe\maven-1.0.2\bin;C:\Perl\bin;C:\WINXP\system32;C:\WINX
INXP\System32\Wbem;C:\Program Files\VideoLAN\VLC;C:\Program Files\apache-a
.5\bin;C:\Program Files\Support Tools\;C:\Program Files\Microsoft Visual S
Common\Tools\WinNT;C:\Program Files\Microsoft Visual Studio\Common\Tools;C
ram Files\Microsoft Visual Studio\VC98\bin;C:\Program Files\CVSNT\;C:\Prog
les\Java\jdk1.5.0_14\bin

C:\GBPackage\wscontext_exe>set JAVA_HOME=C:\Program Files\Java\jdk1.5.0_14
C:\GBPackage\wscontext_exe>set MAVEN_HOME=C:\GBPackage\wscontext_exe\maven

C:\GBPackage\wscontext_exe>set CATALINA_HOME=C:\GBPackage\wscontext_exe\ja
tomcat-5.0.28
C:\GBPackage\wscontext_exe>_
```

4. Execute “startmysql4.0.bat”



```
C:\WINXP\system32\cmd.exe - startmysql4.0.bat
C:\GBPackage\wscontext_exe>env.bat

C:\GBPackage\wscontext_exe>set PATH=C:\Program Files\Java\jdk1.5.0_14\bin;
ackage\wscontext_exe\maven-1.0.2\bin;C:\Perl\bin;C:\WINXP\system32;C:\WINX
INXP\System32\Wbem;C:\Program Files\VideoLAN\VLC;C:\Program Files\apache-a
.5\bin;C:\Program Files\Support Tools\;C:\Program Files\Microsoft Visual S
Common\Tools\WinNT;C:\Program Files\Microsoft Visual Studio\Common\Tools;C
ram Files\Microsoft Visual Studio\VC98\bin;C:\Program Files\CUSNT\;C:\Prog
les\Java\jdk1.5.0_14\bin

C:\GBPackage\wscontext_exe>set JAVA_HOME=C:\Program Files\Java\jdk1.5.0_14
C:\GBPackage\wscontext_exe>set MAVEN_HOME=C:\GBPackage\wscontext_exe\maven

C:\GBPackage\wscontext_exe>set CATALINA_HOME=C:\GBPackage\wscontext_exe\ja
tomcat-5.0.28

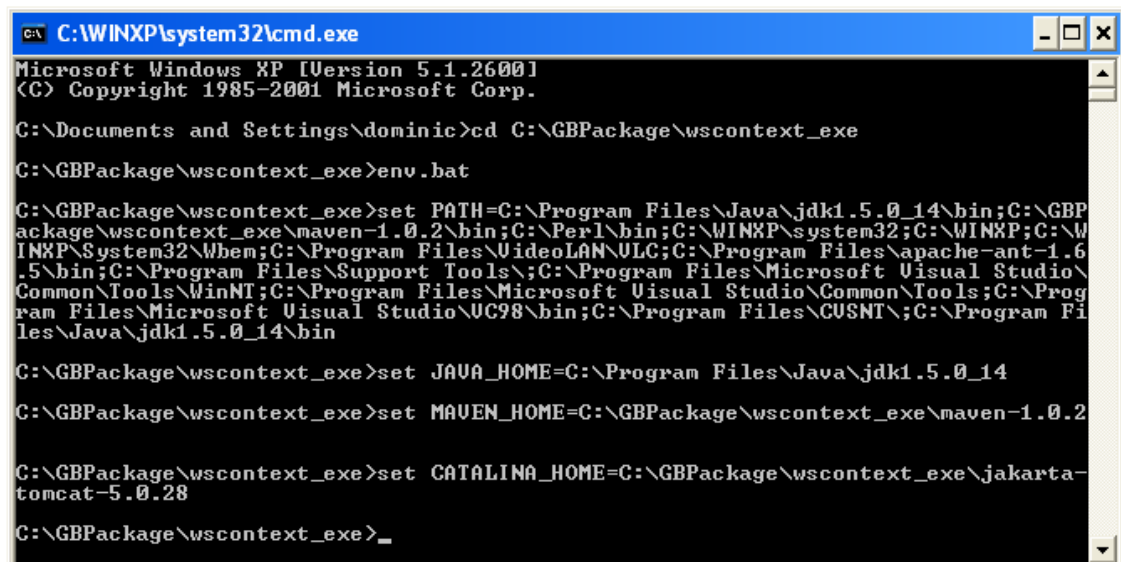
C:\GBPackage\wscontext_exe>startmysql4.0.bat

C:\GBPackage\wscontext_exe>mysql-4.0.27-win32\bin\mysqld-nt --console --sk
odb --basedir=C:\GBPackage\wscontext_exe\mysql-4.0.27-win32
mysql-4.0.27-win32\bin\mysqld-nt: ready for connections.
Version: '4.0.27-nt' socket: '' port: 3306 Official MySQL binary
```

Starting AXIS

Please follow these steps to start AXIS server:

1. Open “cmd” by clicking Windows’ “Start” -> Run -> type “cmd” -> OK
2. Go to <installation directory>\wscontext_exe
3. Execute “env.bat”



```
C:\WINXP\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

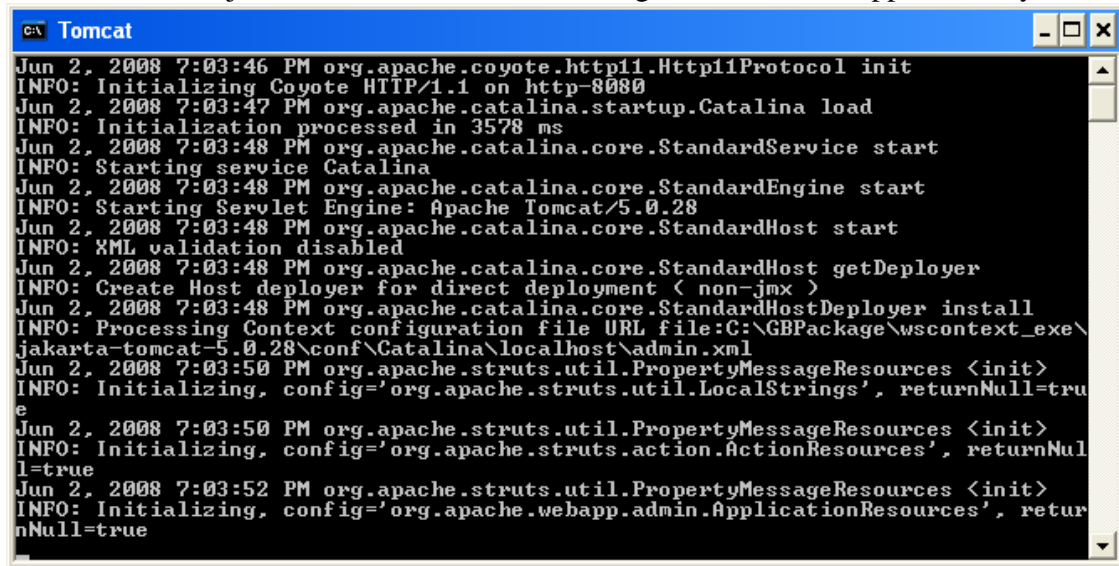
C:\Documents and Settings\dominic>cd C:\GBPackage\wscontext_exe

C:\GBPackage\wscontext_exe>env.bat

C:\GBPackage\wscontext_exe>set PATH=C:\Program Files\Java\jdk1.5.0_14\bin;C:\GBP
ackage\wscontext_exe\maven-1.0.2\bin;C:\Perl\bin;C:\WINXP\system32;C:\WINXP;C:\W
INXP\System32\Wbem;C:\Program Files\VideoLAN\VLC;C:\Program Files\apache-ant-1.6
.5\bin;C:\Program Files\Support Tools\;C:\Program Files\Microsoft Visual Studio\
Common\Tools\WinNT;C:\Program Files\Microsoft Visual Studio\Common\Tools;C:\Prog
ram Files\Microsoft Visual Studio\VC98\bin;C:\Program Files\CUSNT\;C:\Program Fi
les\Java\jdk1.5.0_14\bin

C:\GBPackage\wscontext_exe>set JAVA_HOME=C:\Program Files\Java\jdk1.5.0_14
C:\GBPackage\wscontext_exe>set MAVEN_HOME=C:\GBPackage\wscontext_exe\maven-1.0.2
C:\GBPackage\wscontext_exe>set CATALINA_HOME=C:\GBPackage\wscontext_exe\jakarta-
tomcat-5.0.28
C:\GBPackage\wscontext_exe>_
```


4. Execute “startjakartatomcat.bat”. The following window should appear shortly.



```

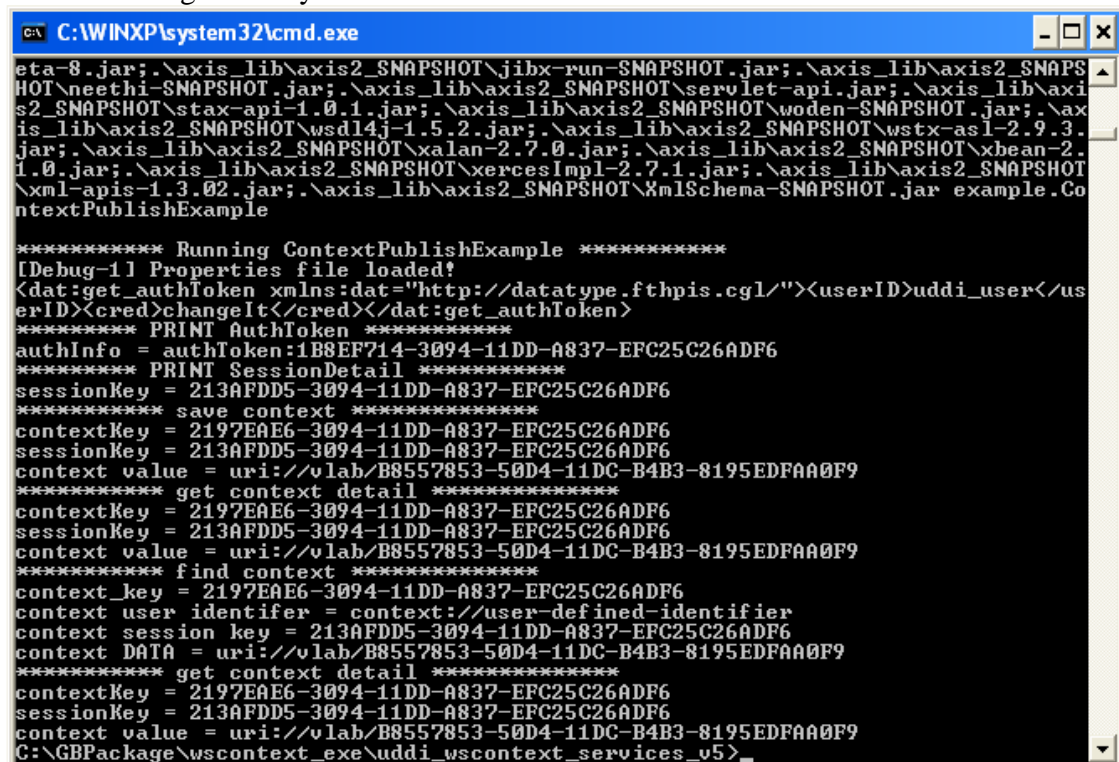
C:\ Tomcat
Jun 2, 2008 7:03:46 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-8080
Jun 2, 2008 7:03:47 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 3578 ms
Jun 2, 2008 7:03:48 PM org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
Jun 2, 2008 7:03:48 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/5.0.28
Jun 2, 2008 7:03:48 PM org.apache.catalina.core.StandardHost start
INFO: XML validation disabled
Jun 2, 2008 7:03:48 PM org.apache.catalina.core.StandardHost getDeployer
INFO: Create Host Deployer for direct deployment ( non-jmx )
Jun 2, 2008 7:03:48 PM org.apache.catalina.core.StandardHostDeployer install
INFO: Processing Context configuration file URL file:C:\GBPackage\wscontext_exe\jakarta-tomcat-5.0.28\conf\Catalina\localhost\admin.xml
Jun 2, 2008 7:03:50 PM org.apache.struts.util.PropertyMessageResources <init>
INFO: Initializing, config='org.apache.struts.util.LocalStrings', returnNull=true
Jun 2, 2008 7:03:50 PM org.apache.struts.util.PropertyMessageResources <init>
INFO: Initializing, config='org.apache.struts.action.ActionResources', returnNull=true
Jun 2, 2008 7:03:52 PM org.apache.struts.util.PropertyMessageResources <init>
INFO: Initializing, config='org.apache.webapp.admin.ApplicationResources', returnNull=true

```

Testing the deployment

Please follow these steps to verify that WS-Context servers are running correctly:

1. Open “cmd” by clicking Windows’ “Start” -> Run -> type “cmd” -> OK
2. Go to <installation directory>\wscontext_exe\uddi_wscontext_services_v5
3. Execute “setpath.bat”
4. Execute “startContextPublishExample.bat”. If you see the following lines, the servers are running correctly



```

C:\WINXP\system32\cmd.exe
eta-8.jar;.axis_lib\axis2_SNAPSHOT\jibx-run-SNAPSHOT.jar;.axis_lib\axis2_SNAPSHOT\neethi-SNAPSHOT.jar;.axis_lib\axis2_SNAPSHOT\servlet-api.jar;.axis_lib\axis2_SNAPSHOT\stax-api-1.0.1.jar;.axis_lib\axis2_SNAPSHOT\woden-SNAPSHOT.jar;.axis_lib\axis2_SNAPSHOT\wsdl4j-1.5.2.jar;.axis_lib\axis2_SNAPSHOT\wstx-asl-2.9.3.jar;.axis_lib\axis2_SNAPSHOT\xalan-2.7.0.jar;.axis_lib\axis2_SNAPSHOT\xbean-2.1.0.jar;.axis_lib\axis2_SNAPSHOT\xercesImpl-2.7.1.jar;.axis_lib\axis2_SNAPSHOT\xml-apis-1.3.02.jar;.axis_lib\axis2_SNAPSHOT\XmlSchema-SNAPSHOT.jar example.ContextPublishExample

***** Running ContextPublishExample *****
[Debug-1] Properties file loaded!
<dat:get_authToken xmlns:dat="http://datatype.fthtps.cgl/"><userID>uddi_user</userID><cred>changeIt</cred></dat:get_authToken>
***** PRINT AuthToken *****
authInfo = authToken:1B8EF714-3094-11DD-A837-EFC25C26ADF6
***** PRINT SessionDetail *****
sessionKey = 213AFDD5-3094-11DD-A837-EFC25C26ADF6
***** save context *****
contextKey = 2197EAE6-3094-11DD-A837-EFC25C26ADF6
sessionKey = 213AFDD5-3094-11DD-A837-EFC25C26ADF6
context value = uri://vlab/B8557853-50D4-11DC-B4B3-8195EDFAA0F9
***** get context detail *****
contextKey = 2197EAE6-3094-11DD-A837-EFC25C26ADF6
sessionKey = 213AFDD5-3094-11DD-A837-EFC25C26ADF6
context value = uri://vlab/B8557853-50D4-11DC-B4B3-8195EDFAA0F9
***** find context *****
context_key = 2197EAE6-3094-11DD-A837-EFC25C26ADF6
context user identifier = context://user-defined-identifier
context session key = 213AFDD5-3094-11DD-A837-EFC25C26ADF6
context DATA = uri://vlab/B8557853-50D4-11DC-B4B3-8195EDFAA0F9
***** get context detail *****
contextKey = 2197EAE6-3094-11DD-A837-EFC25C26ADF6
sessionKey = 213AFDD5-3094-11DD-A837-EFC25C26ADF6
context value = uri://vlab/B8557853-50D4-11DC-B4B3-8195EDFAA0F9
C:\GBPackage\wscontext_exe\uddi_wscontext_services_v5>

```

5. Execute “startSessionPublishExample.bat”. If you see the following lines, the servers are running correctly

```
C:\WINXP\system32\cmd.exe
dSchema.jar;.\\lib\\mapping.jar;.\\lib\\WS-ContextModule.jar;.\\lib\\wscontext.jar;.\\b
uild\\lib\\HYBRID_SERVICE.aar;.\\axis_lib\\axis2_SNAPSHOT\\annogen-0.1.0.jar;.\\axis_l
ib\\axis2_SNAPSHOT\\axiom-api-SNAPSHOT.jar;.\\axis_lib\\axis2_SNAPSHOT\\axiom-dom-SNA
PSHOT.jar;.\\axis_lib\\axis2_SNAPSHOT\\axiom-impl-SNAPSHOT.jar;.\\axis_lib\\axis2_SNA
PSHOT\\axis2-adb-codegen-SNAPSHOT.jar;.\\axis_lib\\axis2_SNAPSHOT\\axis2-adb-SNAPSHO
T.jar;.\\axis_lib\\axis2_SNAPSHOT\\axis2-codegen-SNAPSHOT.jar;.\\axis_lib\\axis2_SNA
PSHOT\\axis2-java2wsdl-SNAPSHOT.jar;.\\axis_lib\\axis2_SNAPSHOT\\axis2-jaxbri-SNAPSHO
T.jar;.\\axis_lib\\axis2_SNAPSHOT\\axis2-jibx-SNAPSHOT.jar;.\\axis_lib\\axis2_SNA
PSHOT\\axis2-kernel-SNAPSHOT.jar;.\\axis_lib\\axis2_SNAPSHOT\\axis2-soapmonitor-SNAPSHOT
.jar;.\\axis_lib\\axis2_SNAPSHOT\\axis2-xmlbeans-SNAPSHOT.jar;.\\axis_lib\\axis2_SNA
PSHOT\\backport-util-concurrent-2.1.jar;.\\axis_lib\\axis2_SNAPSHOT\\commons-codec-1.
3.jar;.\\axis_lib\\axis2_SNAPSHOT\\commons-fileupload-1.0.jar;.\\axis_lib\\axis2_SNA
PSHOT\\commons-httpclient-3.0.jar;.\\axis_lib\\axis2_SNAPSHOT\\commons-logging-1.1.ja
r;.\\axis_lib\\axis2_SNAPSHOT\\geronimo-spec-activation-1.0.2-rc4.jar;.\\axis_lib\\ax
is2_SNAPSHOT\\geronimo-spec-jms-1.1-rc4.jar;.\\axis_lib\\axis2_SNAPSHOT\\jaxen-1.1-b
eta-8.jar;.\\axis_lib\\axis2_SNAPSHOT\\jibx-run-SNAPSHOT.jar;.\\axis_lib\\axis2_SNA
PSHOT\\neethi-SNAPSHOT.jar;.\\axis_lib\\axis2_SNAPSHOT\\servlet-api.jar;.\\axis_lib\\axi
s2_SNAPSHOT\\stax-api-1.0.1.jar;.\\axis_lib\\axis2_SNAPSHOT\\wsdl4j-1.5.2.jar;.\\ax
is_lib\\axis2_SNAPSHOT\\wsdl4j-1.5.2.jar;.\\axis_lib\\axis2_SNAPSHOT\\wstx-asl-2.9.3.
jar;.\\axis_lib\\axis2_SNAPSHOT\\xalan-2.7.0.jar;.\\axis_lib\\axis2_SNAPSHOT\\xbean-2.
1.0.jar;.\\axis_lib\\axis2_SNAPSHOT\\xercesImpl-2.7.1.jar;.\\axis_lib\\axis2_SNA
PSHOT\\xml-apis-1.3.02.jar;.\\axis_lib\\axis2_SNAPSHOT\\xmlSchema-SNAPSHOT.jar example.Se
ssionPublishExample

***** Running SessionPublishExample *****
[Debug-1] Properties file loaded!
<dat:get_authToken xmlns:dat="http://datatype.ftthpis.cgl/"><userID>uddi_user</us
erID><cred>changelt</cred></dat:get_authToken>
auth value = authToken:47E3B087-3094-11DD-A837-EFC25C26ADF6
*****
sessionKey = 482B40D8-3094-11DD-A837-EFC25C26ADF6
*****
found session with session_key = 482B40D8-3094-11DD-A837-EFC25C26ADF6
session user key = session://user1/session1/DENEME
C:\GBPackage\wscontext_exe\uddi_wscontext_services_v5>
```

Step 7 – Starting NB 1.3.2

The next step is to start a Messaging Node for sensor data connection. Please follow the steps below to start the node:

1. Modify file <installation directory>\NaradaBrokering-1.3.2_0.37\bin\startBroker.bat. Please configure it as followed. You only have to pay attention to text highlighted in red.

```
@echo off

set NB_HOME=..
set brokerConfigFile=%NB_HOME%\config\BrokerConfiguration.txt
set serviceConfigFile=%NB_HOME%\config\ServiceConfiguration.txt
set brokerCommunicatorPort=11111

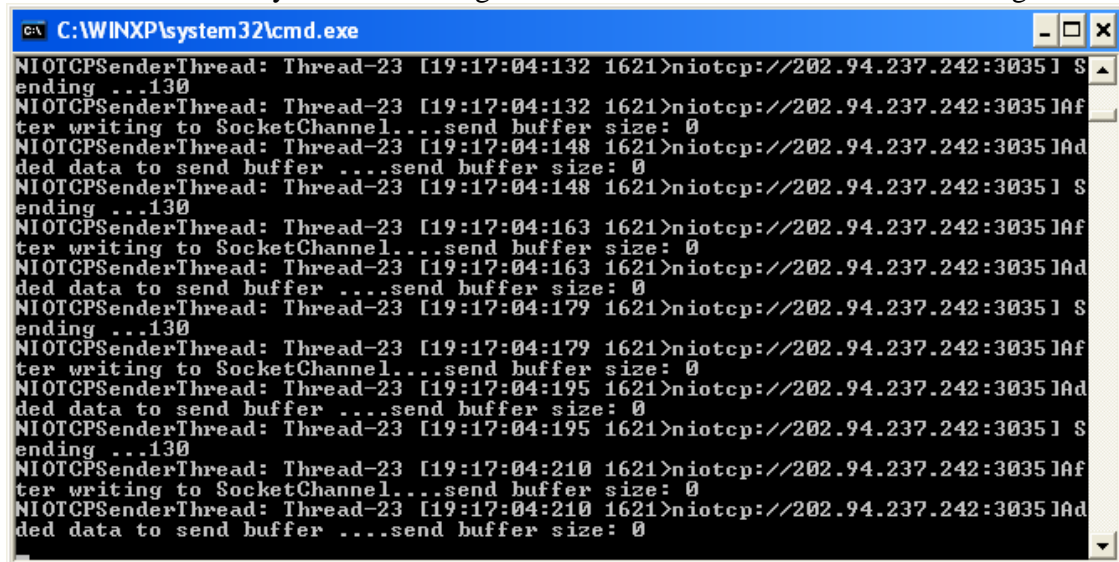
set cp=.
path=%path%;%NB_HOME%\dll

for %i in ("%NB_HOME%\lib\*.jar") do call cpappend.bat %i

java -classpath %cp% cgl.narada.node.BrokerNode %brokerConfigFile%
%serviceConfigFile% %brokerCommunicatorPort% 202.94.237.242 3035
```

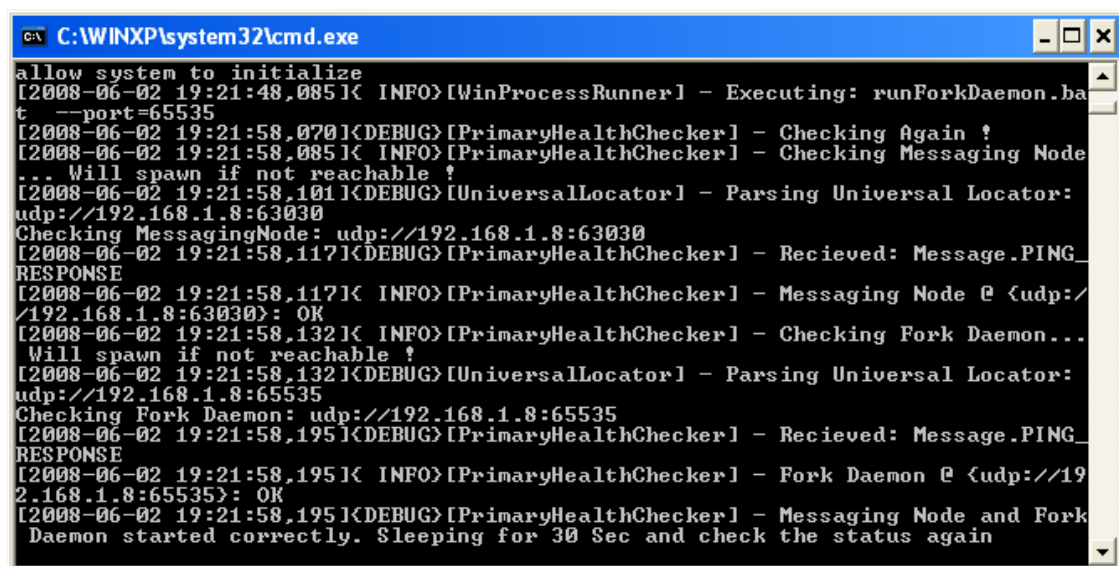
Your installation
directory

2. Execute the file by double-clicking it. You should be able to see the following screen:



Step 8 – Running Primary Health Check

In each domain, go to *<installation directory>\GridBuilder\bin* and execute “runPrimaryHealthCheck.bat”. You should be able to see the following screen:



Before proceeding to the next step, make sure that you can see the sentence “... Sleeping for 30 Sec and check the status again” in ALL domains.

Step 9 – Starting Bootstrap Service in ROOT

At this stage each domain has a Fork Daemon running and waiting for data arrival. Start Bootstrap Service in the ROOT Domain (i.e. /ISAAC) by following the steps below:

1. In *<installation directory>\GridBuilder\bin*, start the BootStrap Console by double-clicking the “bootStrapUI.bat” icon or typing “bootStrapUI.bat” in a command prompt. You should see the BootStrap Console pops up as follows:

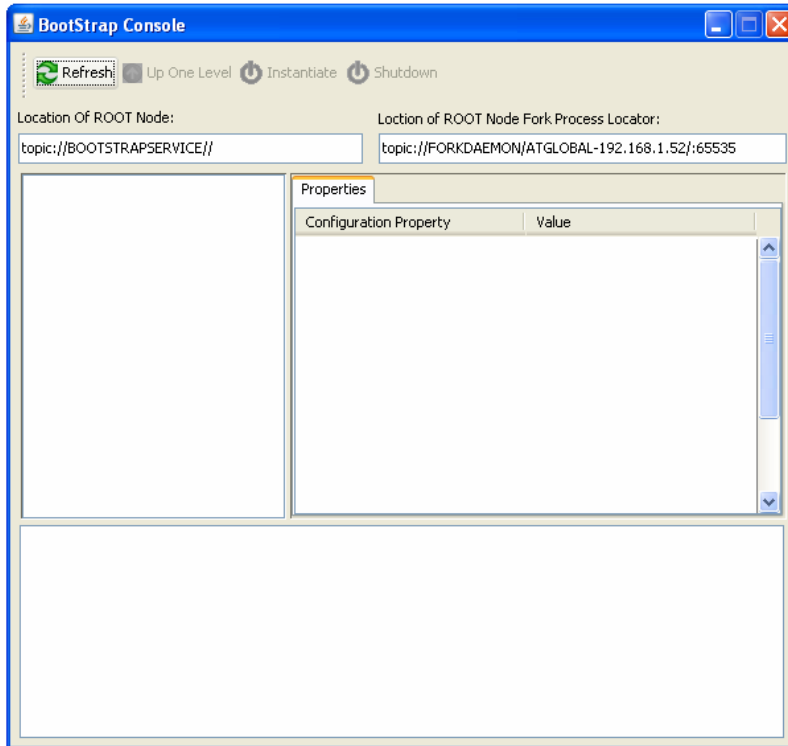


Figure 9-9 User-interface to start the Bootstrap Service in the ROOT domain

2. Click the “Refresh” button. Then, the “Instantiate” button becomes enabled.
3. Click the “Instantiate” button and then click the “Refresh” button again. Now all sub-domains should appear on the left window.

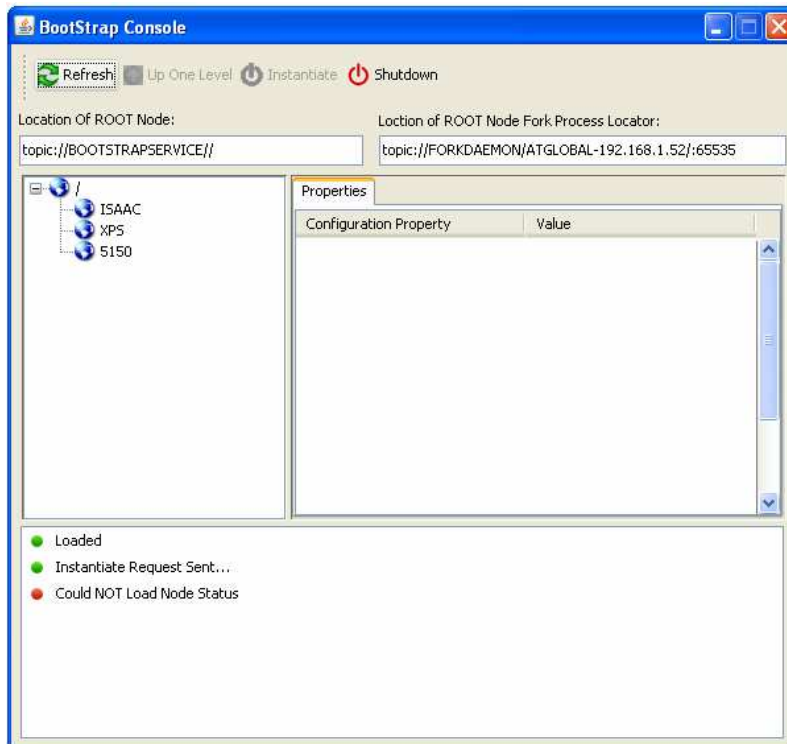


Figure 9-10 A Bootstrap Console showing statuses of a sensor grid sub-domains

Notice that you do NOT have to do the bootstrap process in other domains. Bootstrap processes of sub-domains will be started by ROOT domain automatically.

Step 10 – Starting GB Management Console

Start the GB Management Console (GBMC) in any of the domains by executing *userUI.bat* in <installation directory>\GridBuilder\bin. If all domains are working fine, you should be able to see the screen shown below.

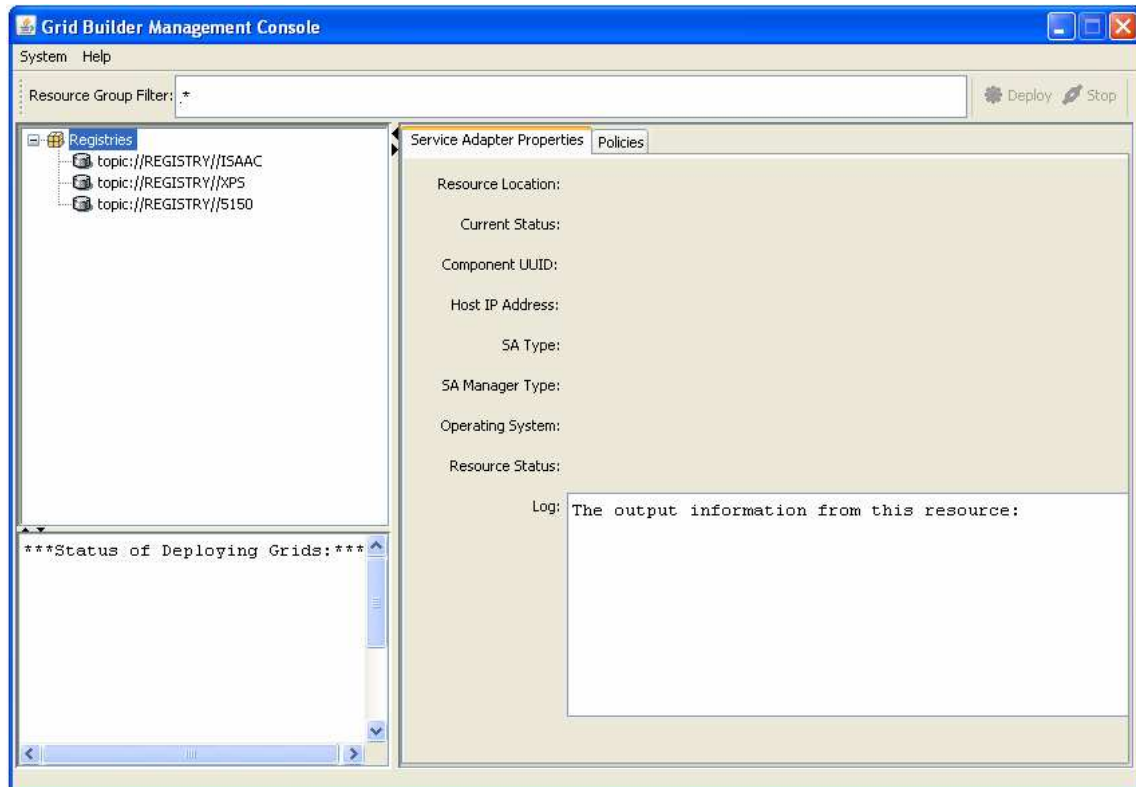


Figure 9-11 An initial view of a Grid Builder Management Console

D.7 Step by Step Sensor Deployment Guide

In this section we are going to deploy 6 types of sensors in the domains, including:

1. Video – a PC Web Camera
2. Video Edge Detection – a Computational Service which takes input from a Video sensor and output processed video including “Edge Detection” and “Region Finding” modes
3. RFID – a RF Code M220 RFID reader which can be connected to a PC through Bluetooth interface. It detects signal strength, tamper, motion and panic information from RFID tags
4. GPS – a satellite positioning device which provides geo-spatial latitude and longitude coordinates. It can be connected to a PC through Bluetooth interface
5. Tribot – a NXT Tribot Lego robot carrying Light, Touch, Sound and Ultrasonic sensors. It can be connected to a PC through Bluetooth interface
6. Wii Remote – a remote controller of the Wii game console. It can be connected to a PC through Bluetooth interface

D.7.1 Deploying a RFID Reader

Please follow these steps to deploy a RF Code M220 RFID reader (rfid1) in one of the Leaf Domains. This RFID reader can be connected to a PC using Bluetooth interface.

1. Before using GBMC, pair up and connects the RFID reader to the PC of the target domain. On the PC, right-click the Bluetooth icon and select “Bluetooth Setup Wizard”



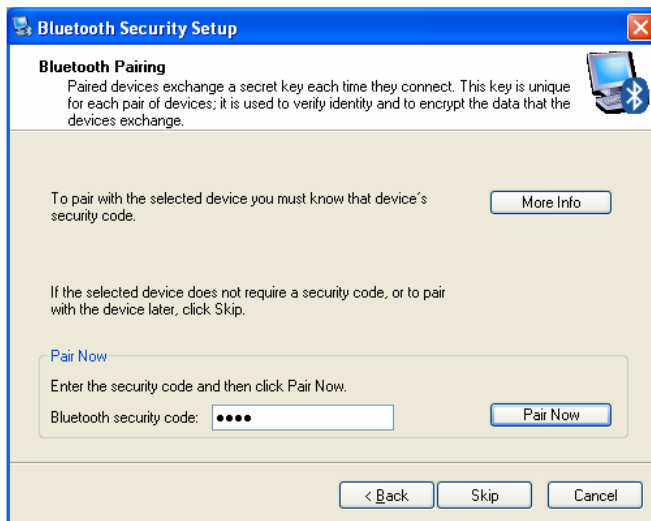
2. Configure a device



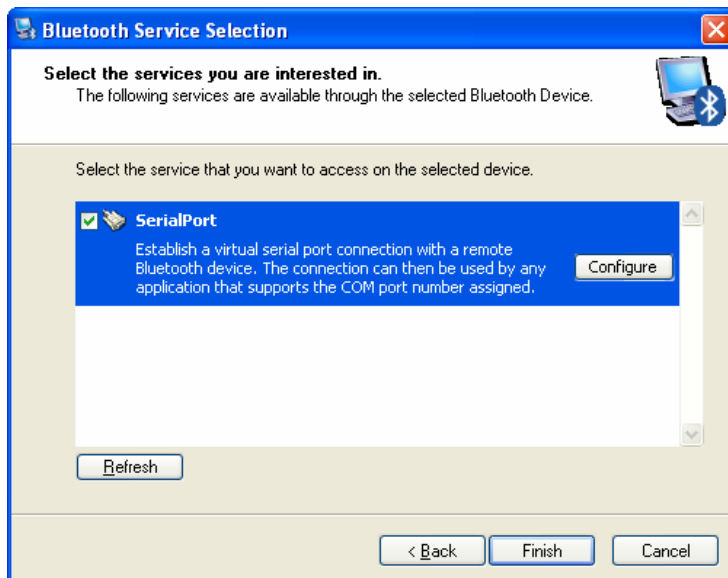
3. Select the RFID reader and click next



4. Pair this RFID reader with its passkey



5. Select “Serial Port” service



6. A window should be popped up to configure this service. Choose a COM port and remember it for later use.



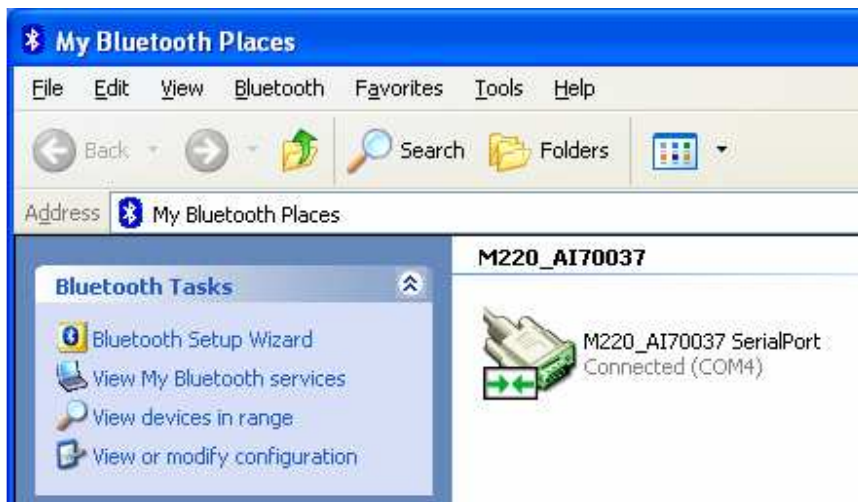
7. Right-click the Bluetooth icon again and select “Explore My Bluetooth Places” this time.



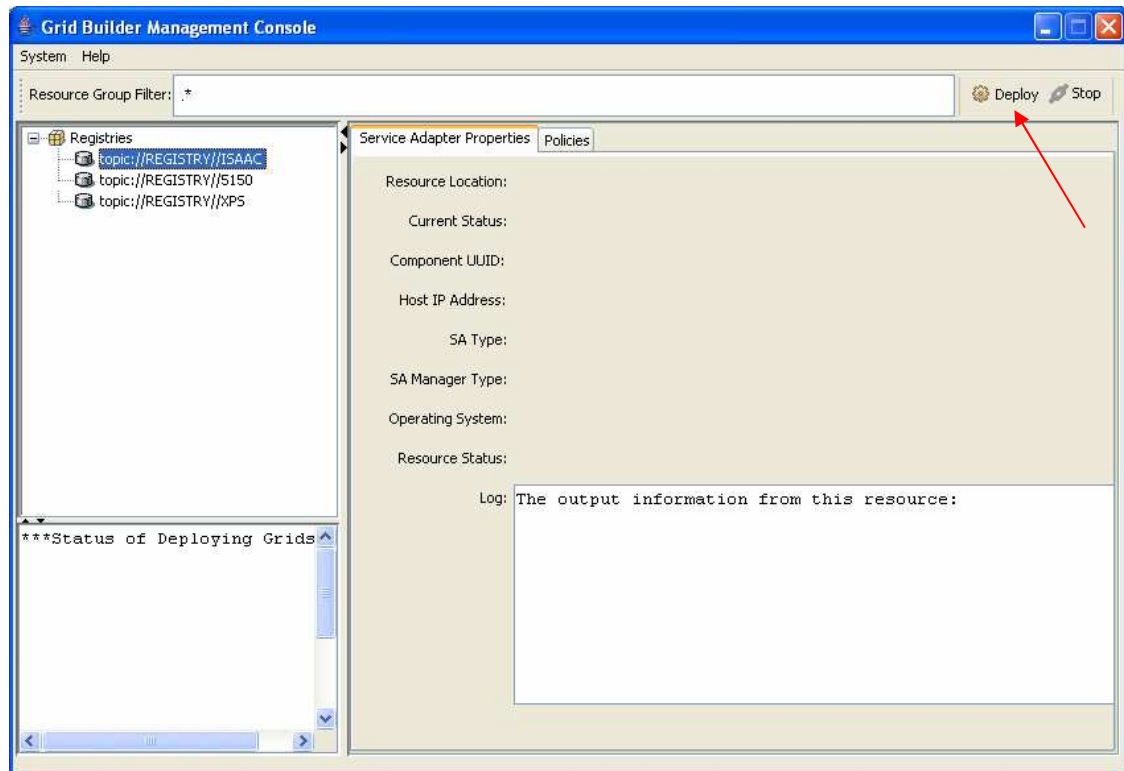
8. Double click the RFID reader



9. If this reader is connected successfully, the following status should be shown. The RFID reader is now connected to COM4.



10. Now go back to GBMC. GBMC shows all leaf domains connected to a particular server. From any PC where GBMC is opened, you can deploy sensors remotely in any domains shown on the left hand side. To start deploying sensors, click on one of the domains on the left hand side, and click “Deploy”.



11. A window should pop up. Enter details of the sensor. For “Sensor Type”, choose RFID. Enter the COM port which the RFID reader is connected. Press “OK” to proceed.

Sensor Selection

Sensor ID: RFID M220

Group ID: demo

Street: Tat Chee Av

City: Hong Kong

State/Province: Hong Kong

Country: Hong Kong

Sensor Type

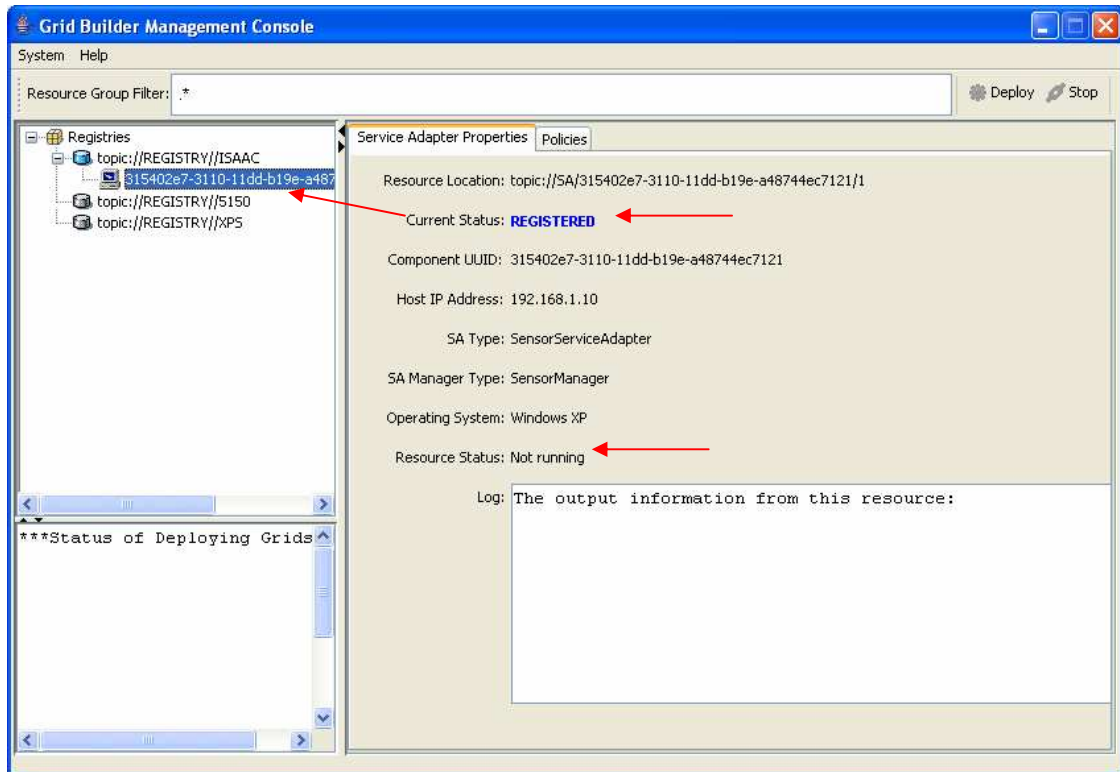
RFID

COM Port

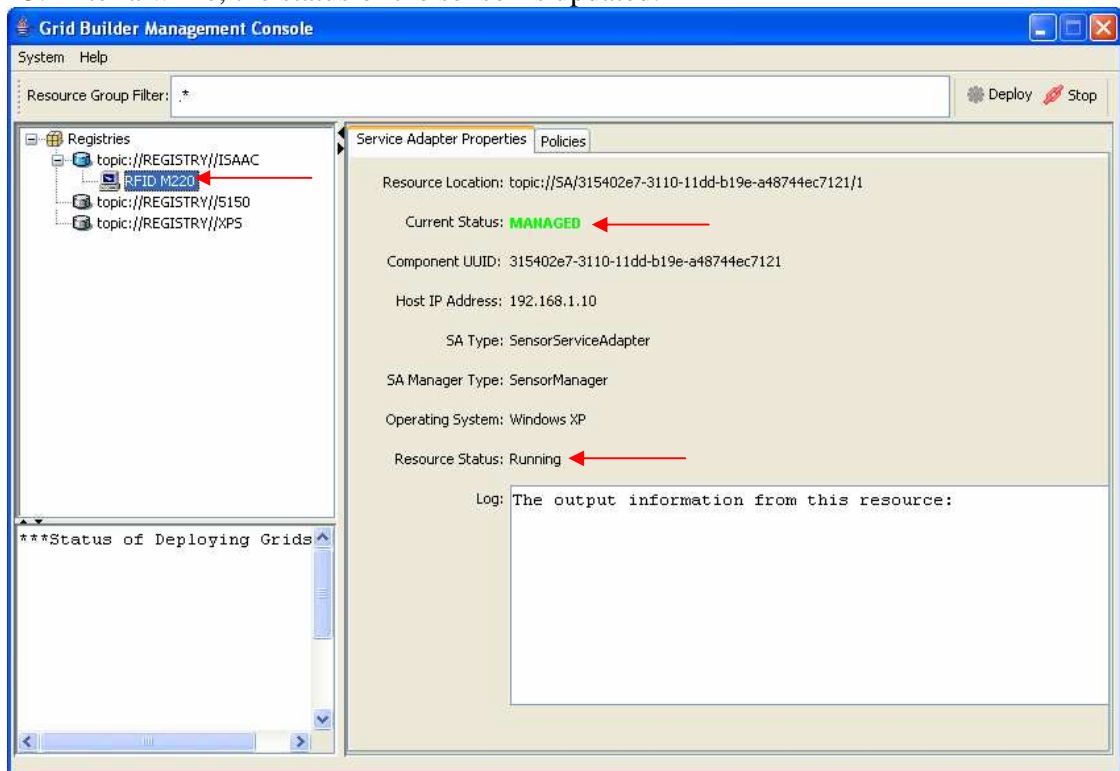
4 (e.g. 1)

OK Cancel

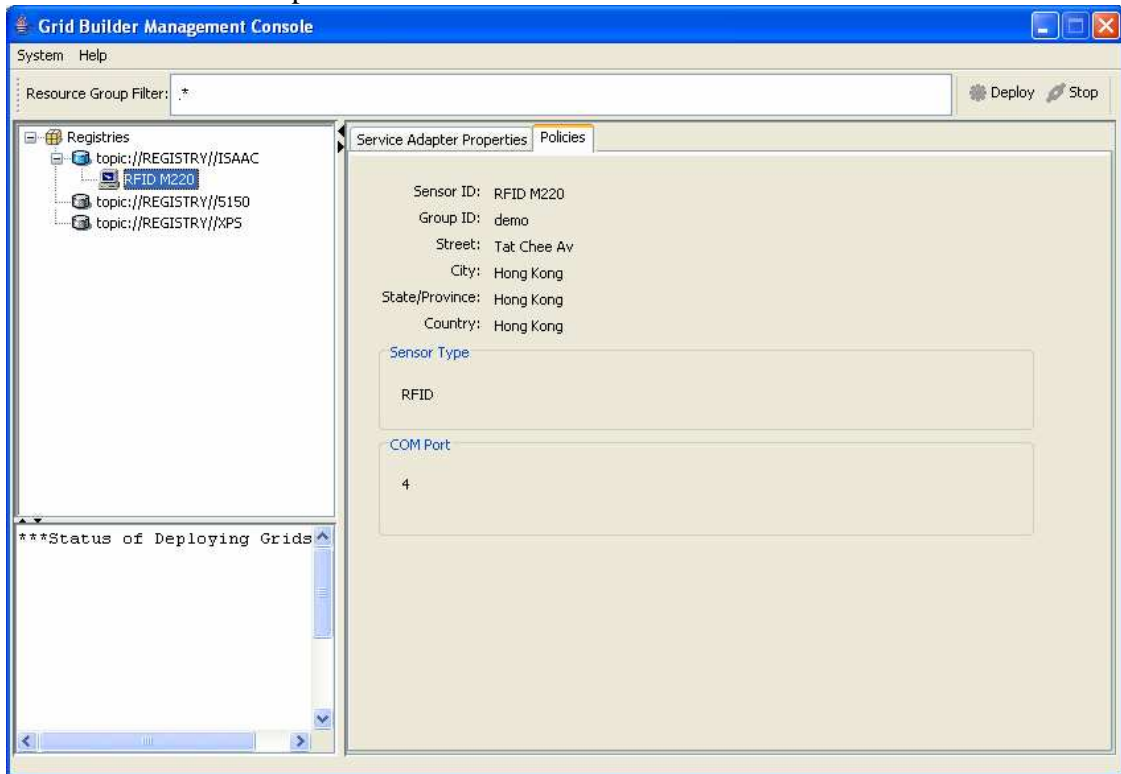
12. After a while you should be able to see a new sensor being deployed in the domain. Notice the “Current Status” and “Resource Status” in the tab “Service Adapter Properties”. “Current Status” shows whether the sensor is being managed by GB. In normal circumstances, “Current Status” should change from REGISTERED to MANAGED within one minute. “Resource Status” shows whether the sensor client program is running. It should change from “Not Running” to “Running” within one minute.



13. After a while, the status of the sensor is updated.



14. You can check the policies of the sensor in tab “Policies”.



D.7.2 Deploying a NXT Tribot Robot

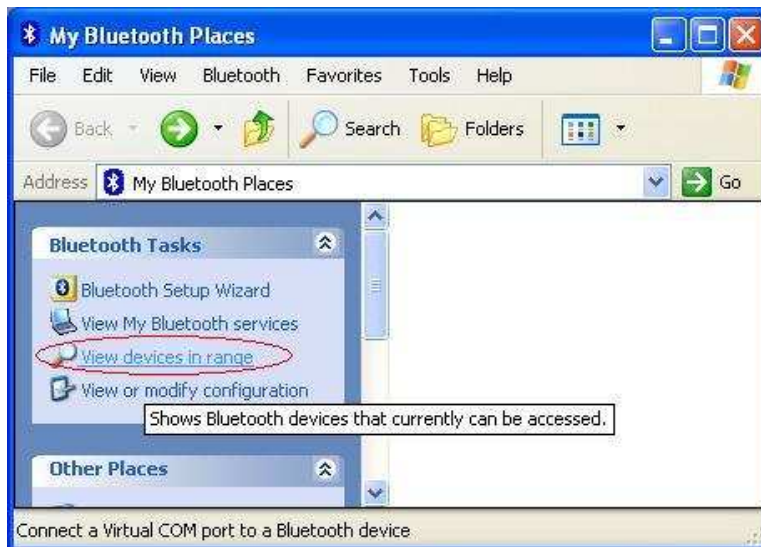
Please follow these steps to deploy a NXT Tribot Robot (Tribot) in one of the Leaf Domains. It can be connected to a PC using Bluetooth interface.

Before using GBMC, find out the Bluetooth address of the robot by following these steps:

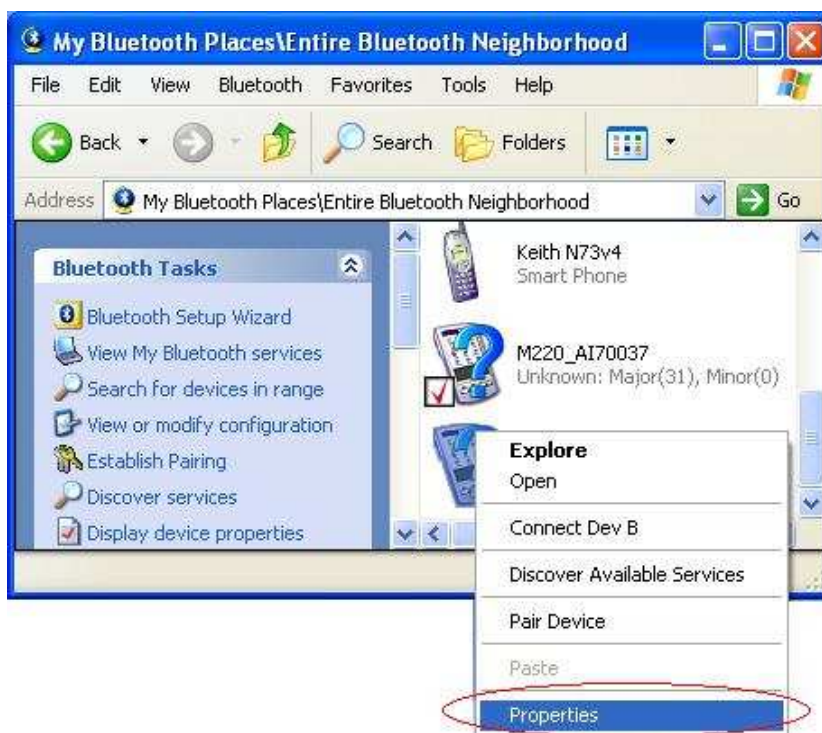
1. Right-click on the Bluetooth icon on the Windows taskbar and choose “Explore my Bluetooth place”.



2. On the left-column of the popup window, choose “View devices in range”.



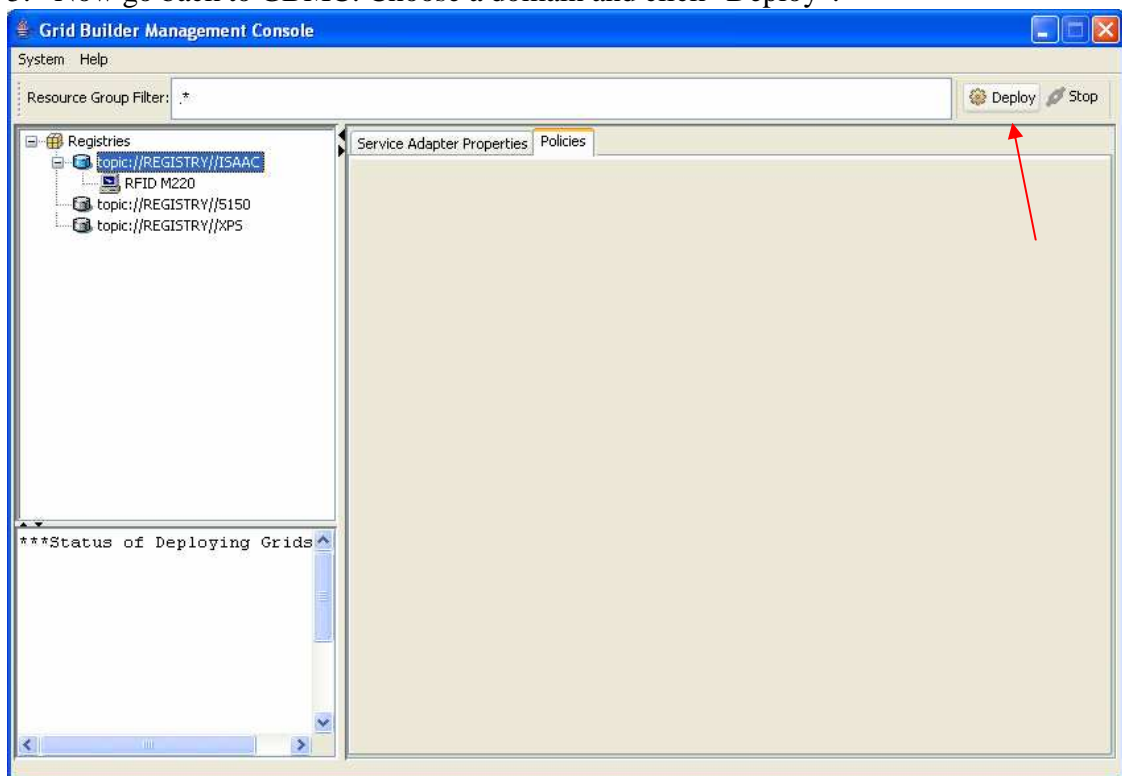
3. Right-click on the device named “NXT” and chooses “Properties”. If you cannot see its presence, make sure that the robot is turned on and choose “Search for devices in range”.



4. The Bluetooth address can be found in the popup window. Write it down on a piece of paper.



5. Now go back to GBMC. Choose a domain and click “Deploy”.



6. Enter the following details. You should enter the Bluetooth address which you have found and written down in step 4. Click “OK”.

Sensor Selection

Sensor ID:

Group ID:

Street:

City:

State/Province:

Country:

Sensor Type:

Robot Sensor Type:

Port 1:

Port 2:

Port 3:

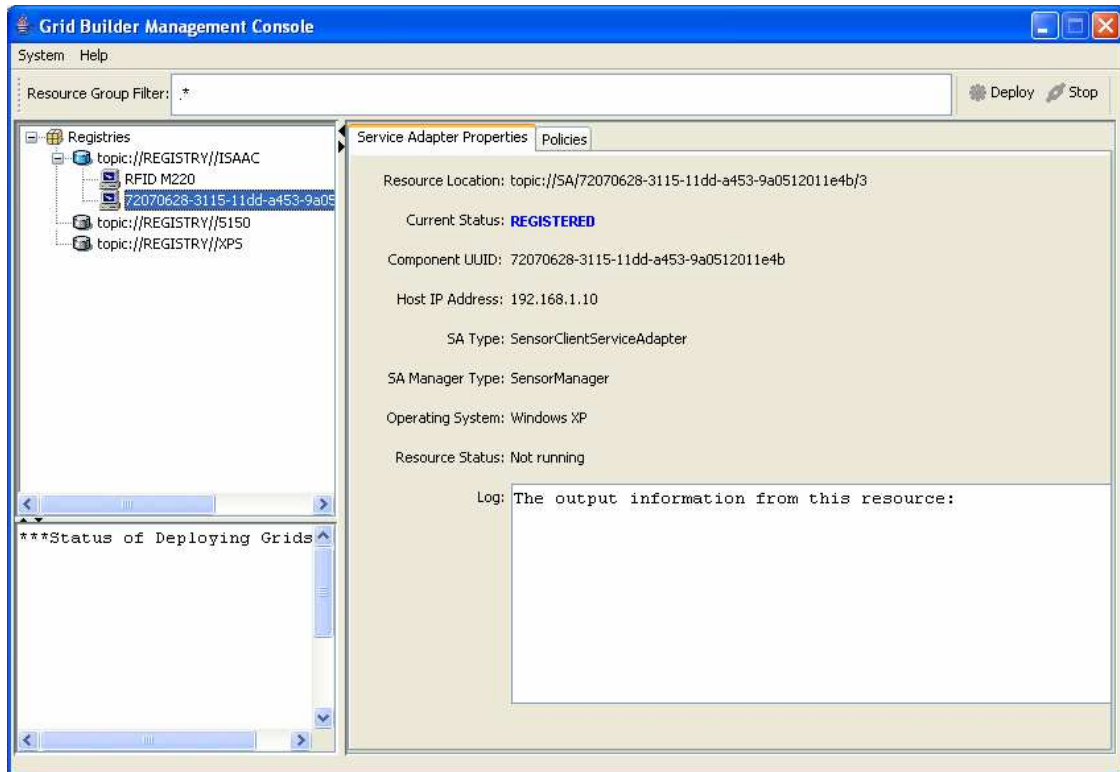
Port 4:

Robot Type:

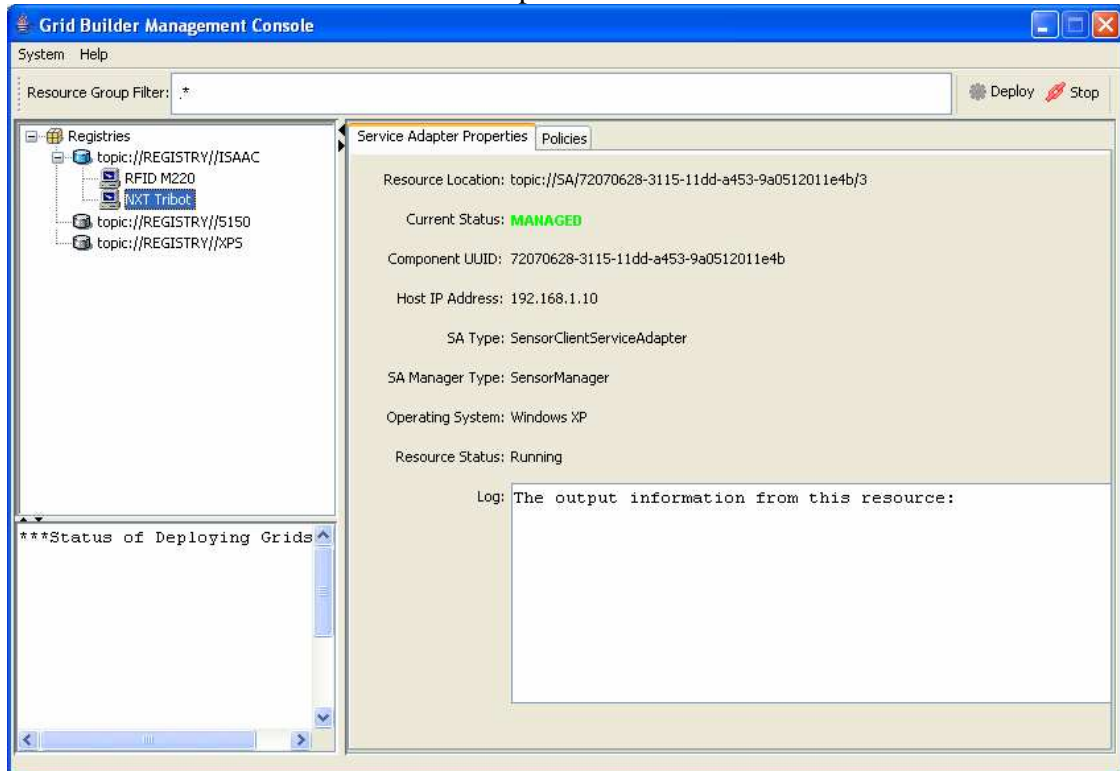
Bluetooth address: (e.g. 00165302ea7c)

OK Cancel

7. You should be able to see the newly deployed robot shortly.



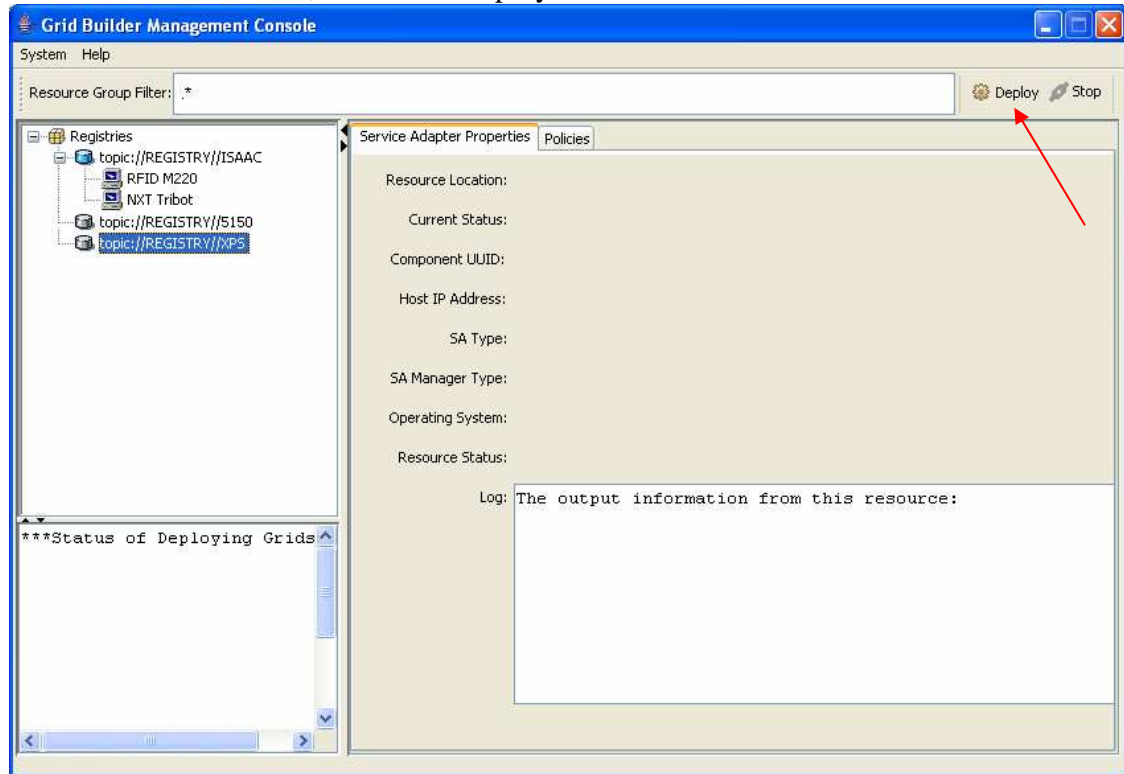
8. After the while the status should be updated.



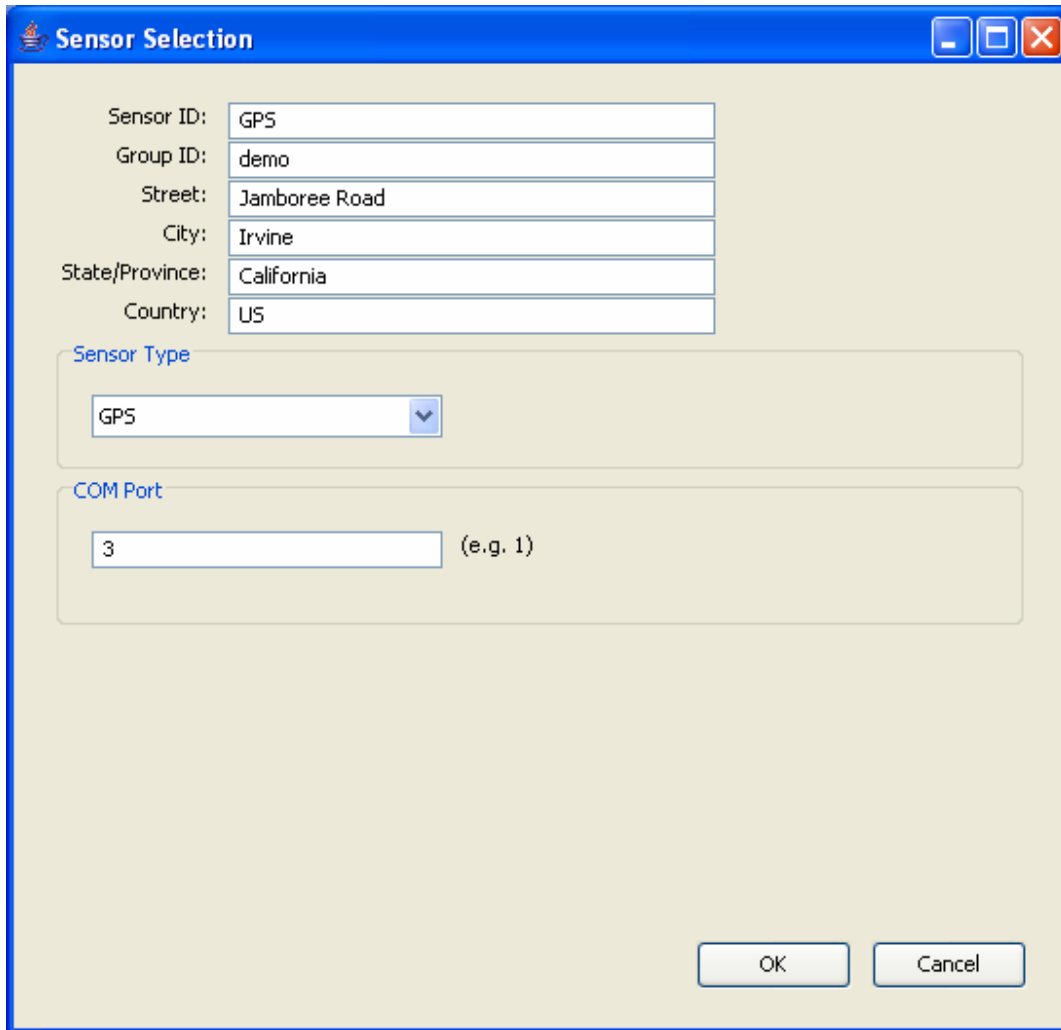
D.7.3 Deploying a GPS

Please follow these steps to deploy aGPS in one of the Leaf Domains. This GPS can be connected to a PC using Bluetooth interface.

1. Pair up and connects the GPS to the PC with similar techniques described in section D.7.1 for RFID deployment.
2. Assume that the GPS is connected to COM3. In GBMC, click on one of the domains on the left hand side, and click “Deploy”.



3. Enter details of the sensor. For “Sensor Type”, choose GPS. Press “OK” to proceed.



The image shows a Windows-style dialog box titled "Sensor Selection". It has a blue title bar with standard minimize, maximize, and close buttons. The main area is light beige and contains several input fields. At the top, there are six text boxes labeled "Sensor ID:", "Group ID:", "Street:", "City:", "State/Province:", and "Country:". These contain the values "GPS", "demo", "Jamboree Road", "Irvine", "California", and "US" respectively. Below these is a section titled "Sensor Type" in blue text, containing a dropdown menu with "GPS" selected. Underneath is another section titled "COM Port" in blue text, containing a text box with the number "3" and a label "(e.g. 1)". At the bottom right, there are two buttons: "OK" and "Cancel".

Sensor ID:	GPS
Group ID:	demo
Street:	Jamboree Road
City:	Irvine
State/Province:	California
Country:	US

Sensor Type

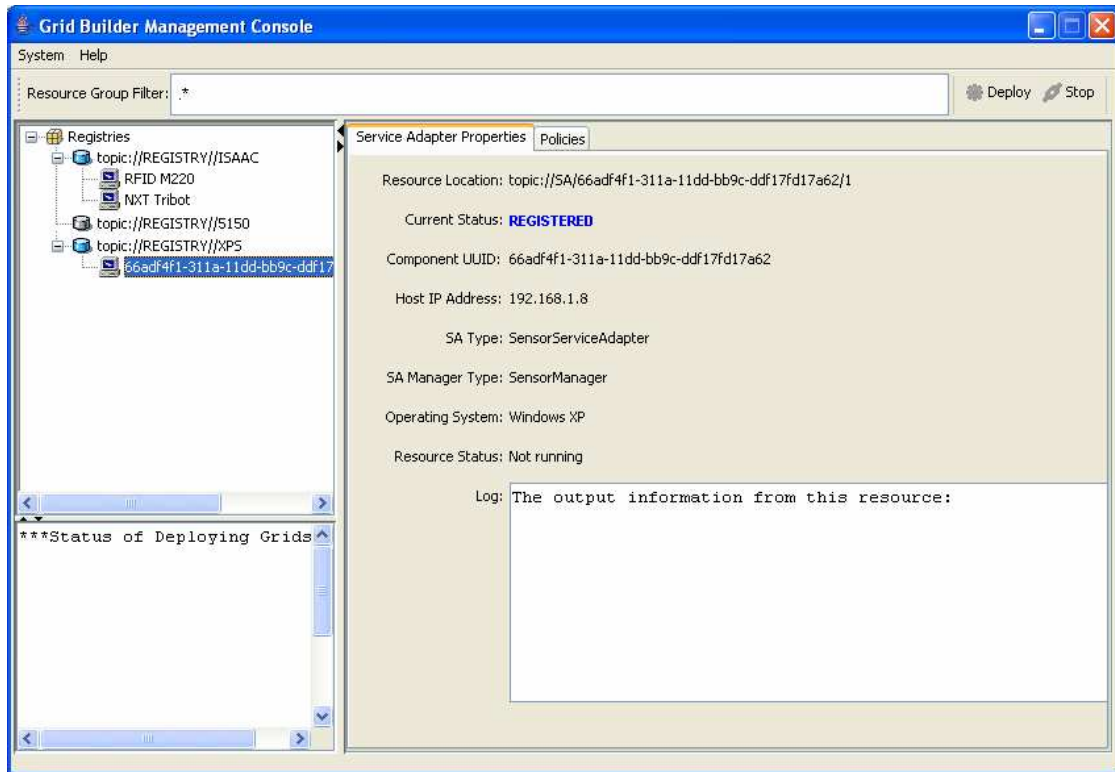
GPS

COM Port

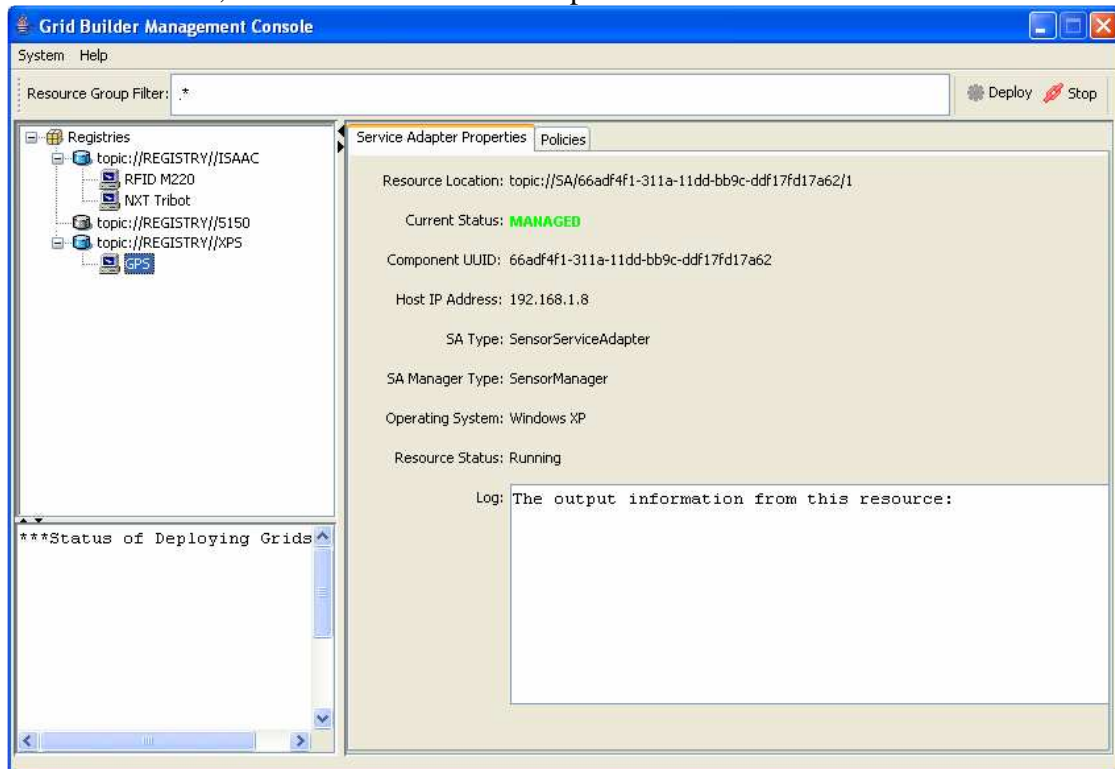
3 (e.g. 1)

OK Cancel

4. After a while you should be able to see a new sensor being deployed in the domain. Notice the "Current Status" and "Resource Status" in the tab "Service Adapter Properties". "Current Status" shows whether the sensor is being managed by GB. In normal circumstances, "Current Status" should change from REGISTERED to MANAGED within one minute. "Resource Status" shows whether the sensor client program is running. It should change from "Not Running" to "Running" within one minute.



5. After a while, the status of the sensor is updated.



D.7.4 Deploying a Wii Remote

Please follow these steps to deploy a Wii Remote in one of the Leaf Domains. This Wii Remote can be connected to a PC using Bluetooth interface.

1. Make sure that a Wii Remote can be detected by a Bluetooth adapter connected to one of the sub-domain computers.
2. On the Console, select the domain which the Wii Remote will be deployed on and then click the “Deploy” button.
3. Enter details of the sensor. For “Sensor Type”, choose “Wii Remote”. Leave the field “Bluetooth Address” blank. Press “OK” to proceed.

Sensor Selection

Sensor ID:

Group ID:

Street:

City:

State/Province:

Country:

Sensor Type

▼

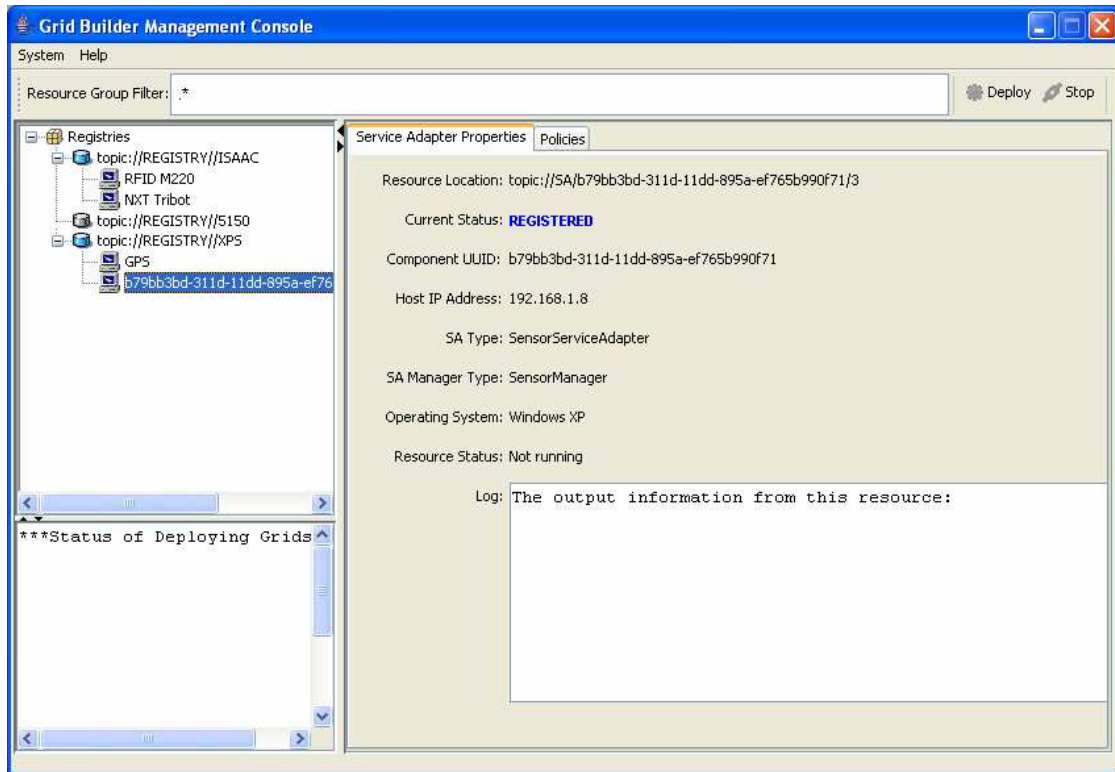
Bluetooth Address

(e.g. 00165302ea7c)

Leave Blank

OK Cancel

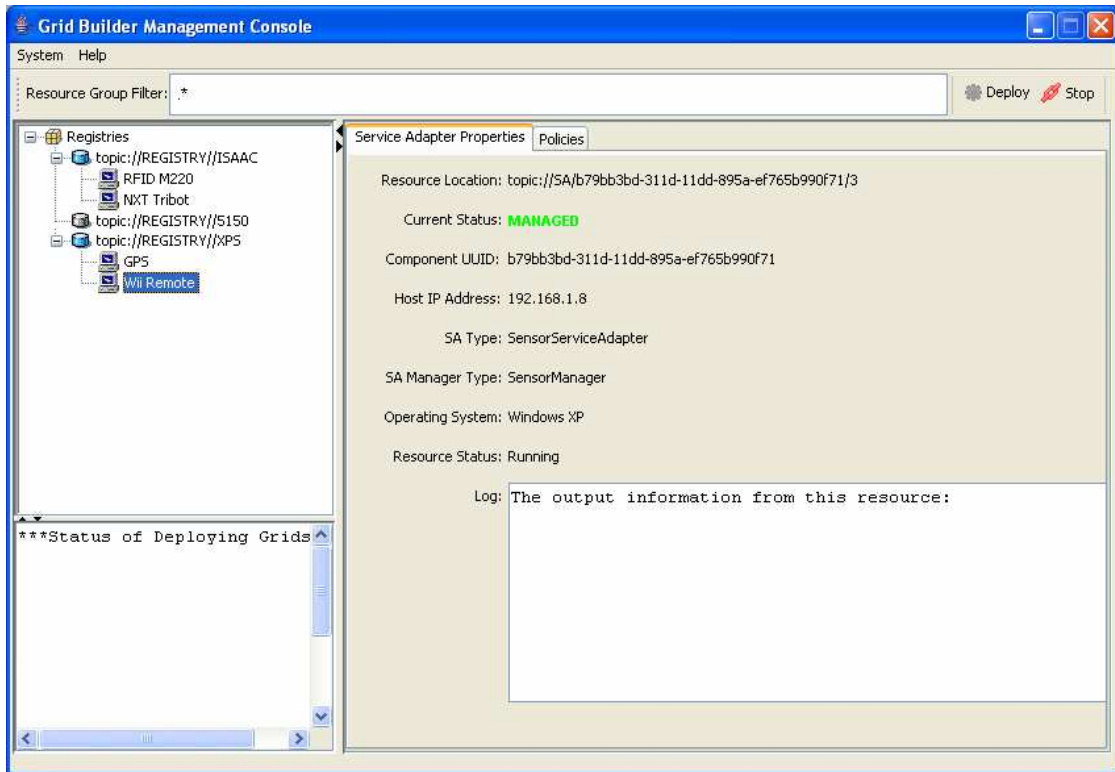
4. Now, a new item appears under the domain currently selected. Select the item to see its service adapter properties. On the Service Adapter Properties page, you should see that Current Status is REGISTERED and Resource Status is Not Running.



5. Press button 1 and 2 of your Wii remote control and hold on. Wait for around 10-20 seconds until the 4 blue light spots stop flashing and only the leftmost one lights up, which means the deployment is successful.



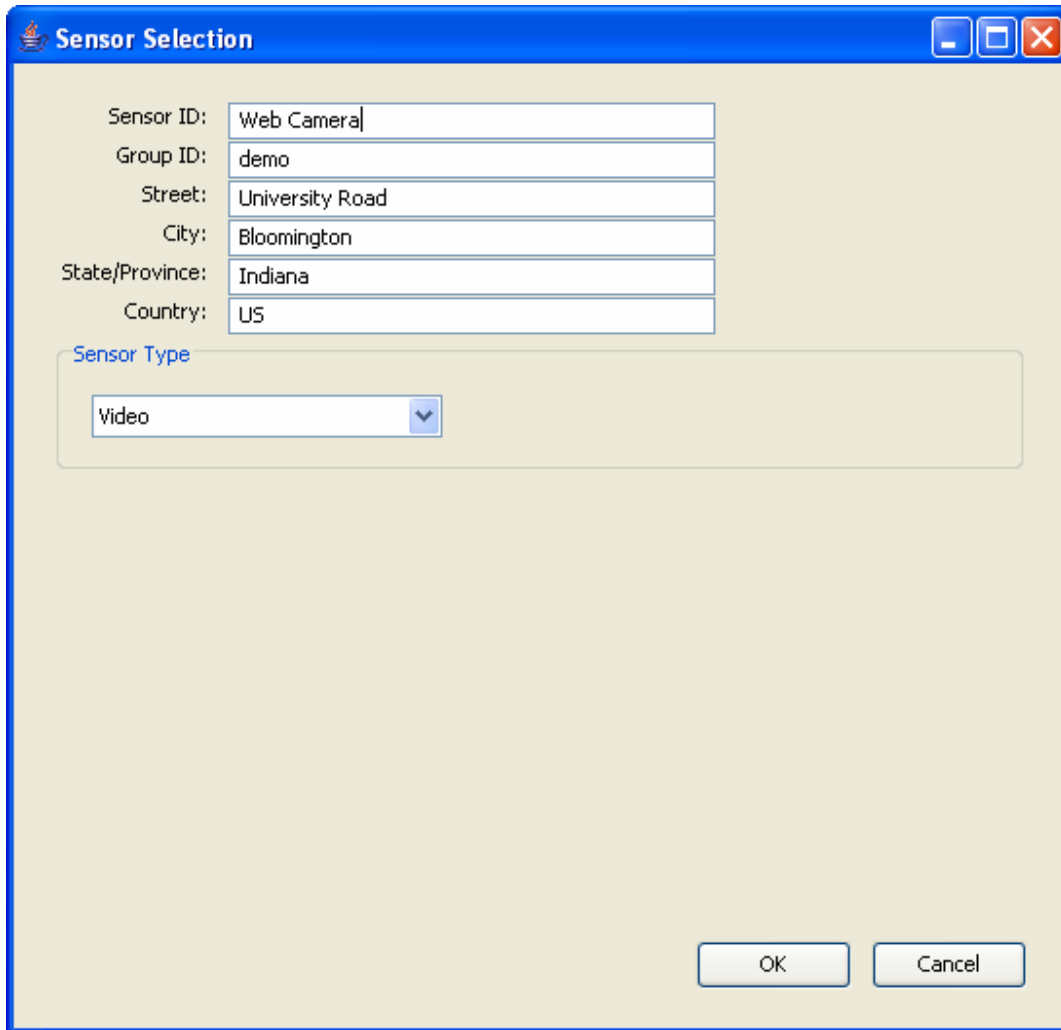
6. Shortly you can see the Wii Remote being deployed successfully.



D.7.5 Deploying a Web Camera

Please follow these steps to deploy a Web Camera in one of the Leaf Domains. This Web Camera can be connected to a PC using the USB interface.

1. Make sure that a webcam is connected to one of the sub-domain computers.
2. On the Console, select the domain which the Web Camera will be deployed on and then click the “Deploy” button.
3. Enter details of the sensor. For “Sensor Type”, choose “Video”. Press “OK” to proceed.



The image shows a Windows-style dialog box titled "Sensor Selection". It contains several text input fields for sensor information: "Sensor ID" (Web Camera), "Group ID" (demo), "Street" (University Road), "City" (Bloomington), "State/Province" (Indiana), and "Country" (US). Below these is a section labeled "Sensor Type" containing a dropdown menu currently set to "Video". At the bottom right are "OK" and "Cancel" buttons.

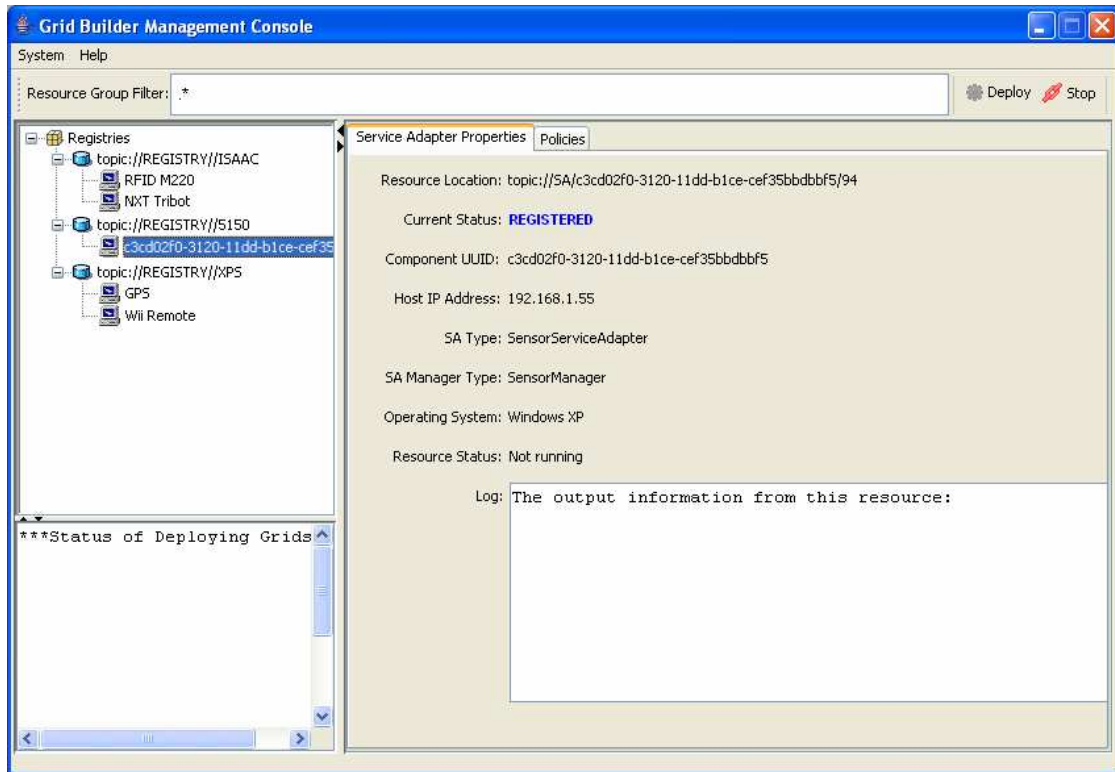
Sensor ID:	Web Camera
Group ID:	demo
Street:	University Road
City:	Bloomington
State/Province:	Indiana
Country:	US

Sensor Type

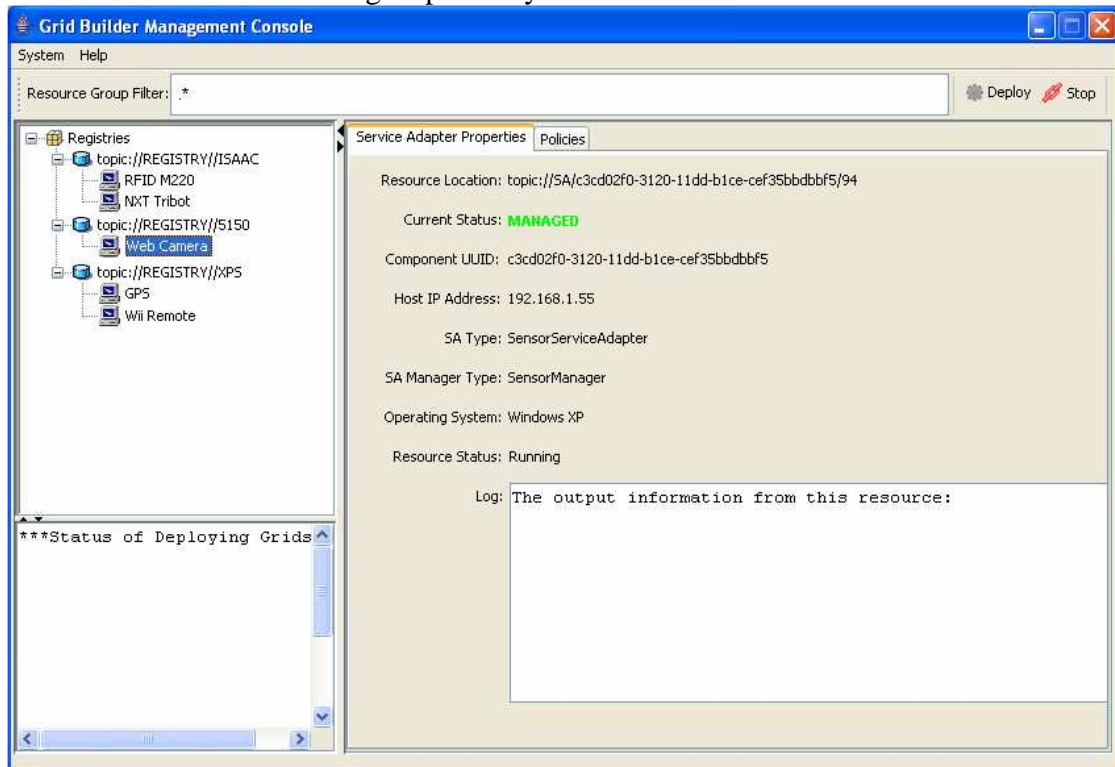
Video

OK Cancel

4. Now, a new item appears under the domain currently selected. Select the item to see its service adapter properties. On the Service Adapter Properties page, you should see that Current Status is REGISTERED and Resource Status is Not Running.



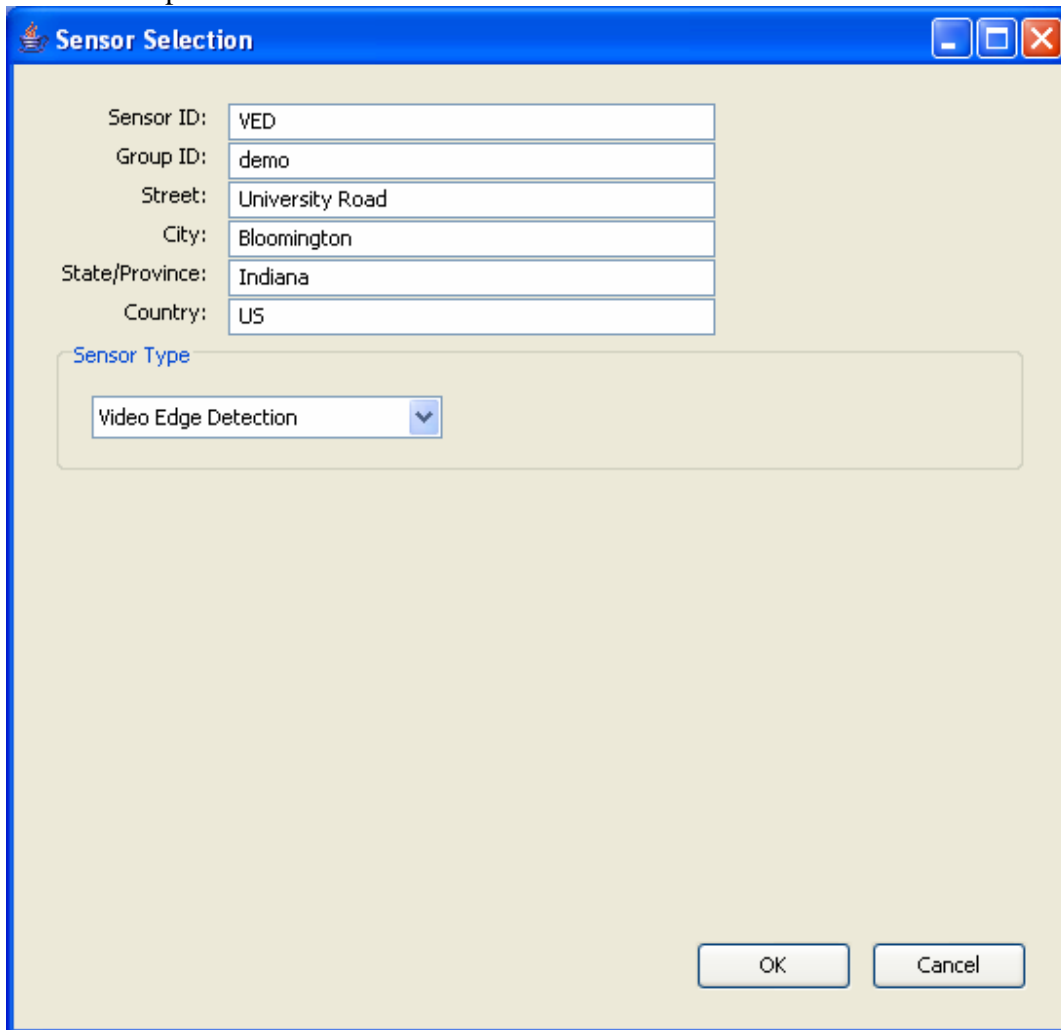
5. If the deployment is successful, Current Status and Resource Status change to **MANAGED** and **Running** respectively after 10-20 seconds.



D.7.6 Deploying a Video Edge Detection Service

Please follow these steps to deploy a Video Edge Detection (VED) service in one of the Leaf Domains. This is a Computational Service and can be deployed in any domain

1. On the Console, select the domain which the VED will be deployed on and then click the “Deploy” button.
2. Enter details of the sensor. For “Sensor Type”, choose “Video Edge Detection”. Press “OK” to proceed.



Sensor Selection

Sensor ID: VED

Group ID: demo

Street: University Road

City: Bloomington

State/Province: Indiana

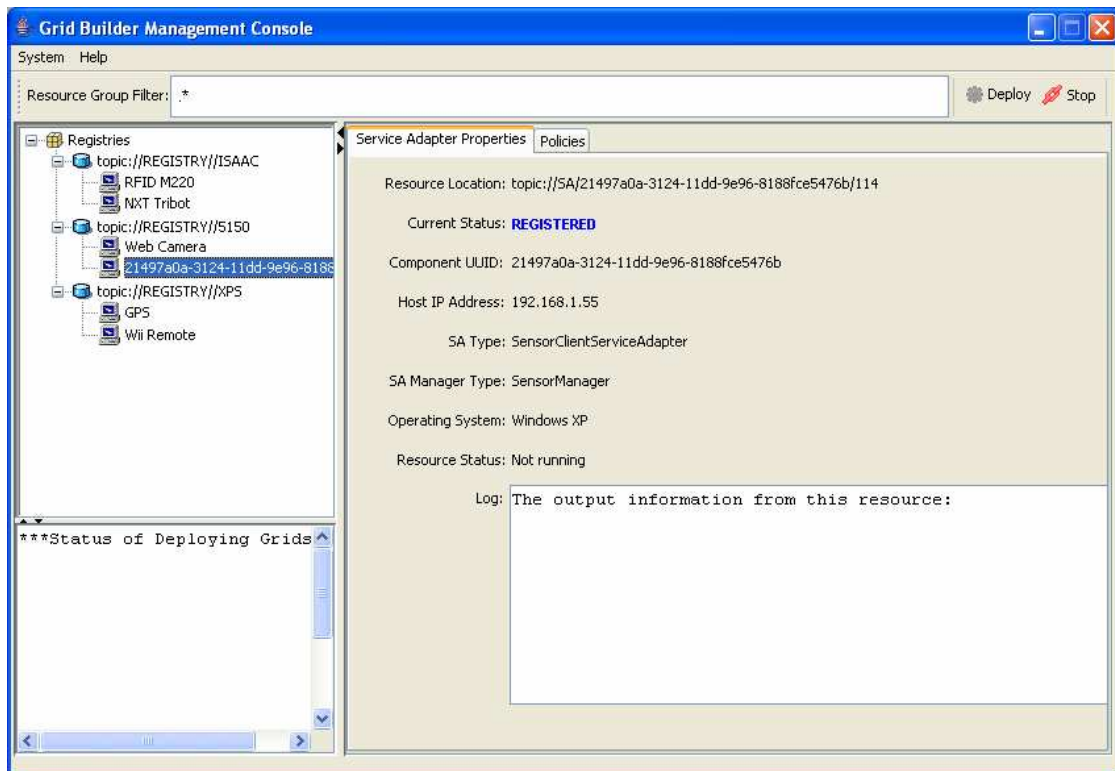
Country: US

Sensor Type

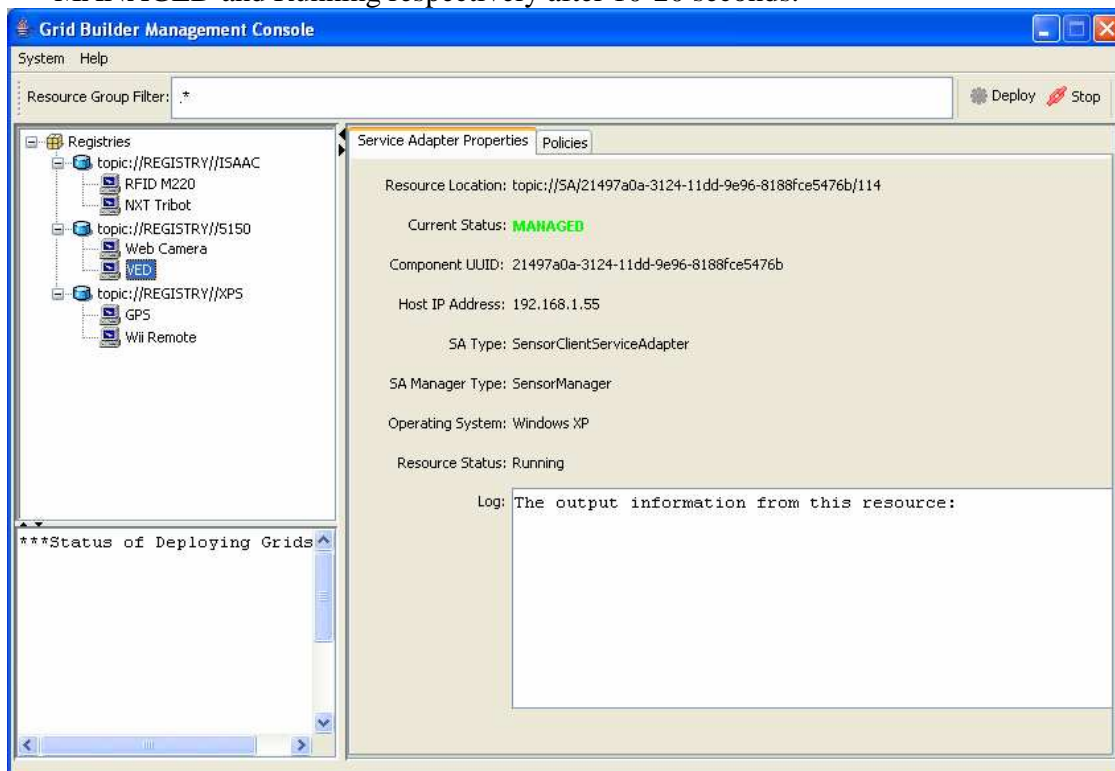
Video Edge Detection

OK Cancel

3. Now, a new item appears under the domain currently selected. Select the item to see its service adapter properties. On the Service Adapter Properties page, you should see that Current Status is REGISTERED and Resource Status is Not Running.



4. If the deployment is successful, Current Status and Resource Status change to **MANAGED** and **Running** respectively after 10-20 seconds.



Appendix E - User Guide for SCGMMS Application User

Impromptu is a framework for developing collaborative applications. Separate applications can be plugged into Impromptu as “Sharedlets”. In order to demonstrate the power of SCGMMS, we have developed a “Sensor Sharedlet”. Sensor Sharedlet is a UDOP application which is defined on an operational environment of sensors. It aims at demonstrating the ability of SCGMMS to support UDOP development by allowing the user to do filtering, visualization and sharing information acquired from a grid of sensors deployed in SCGMMS. The Sensor Sharedlet is labeled as Robot Demo in Figure 9-12.

E.1 Sensor Sharedlet

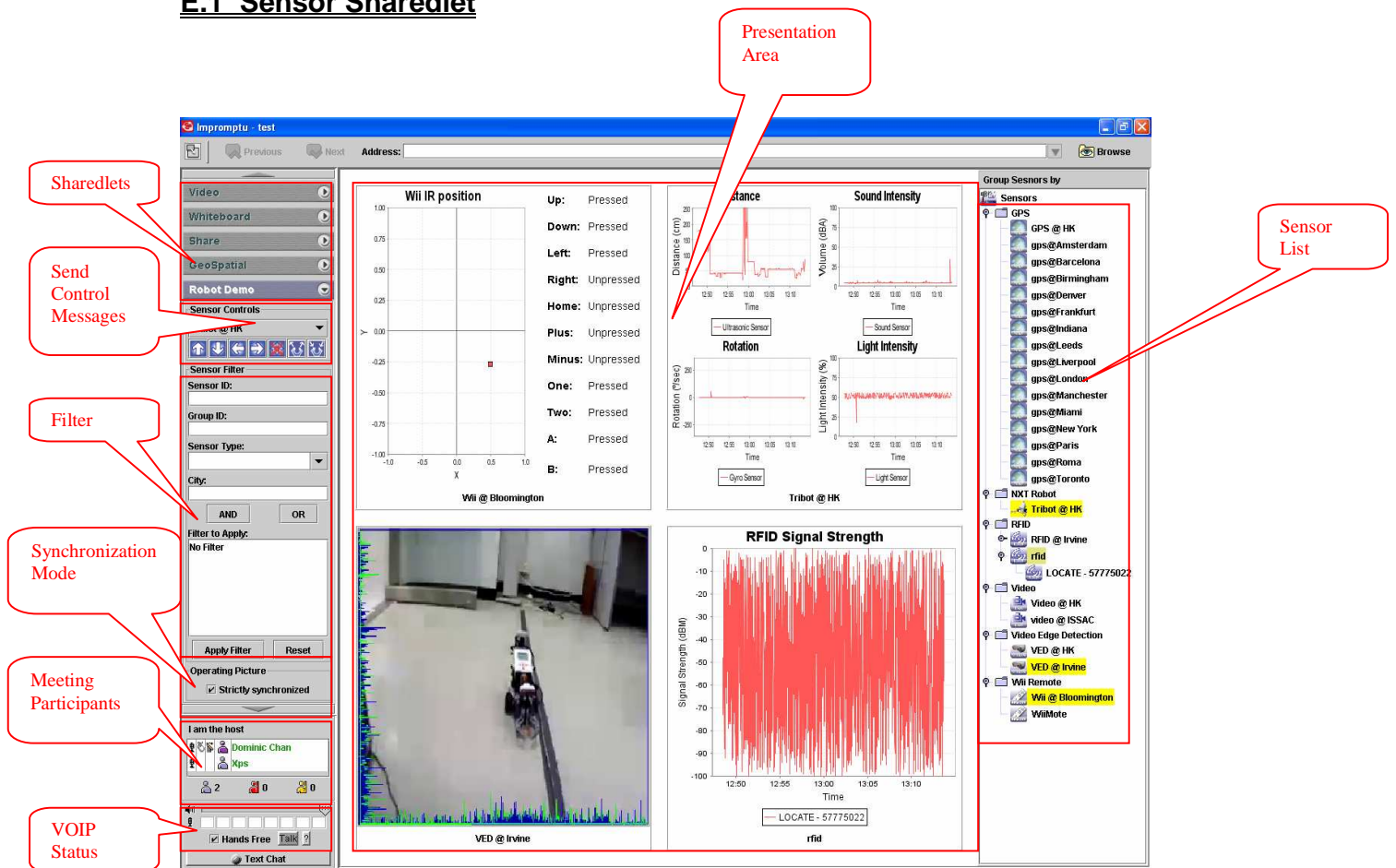


Figure 9-12 Overview of Sensor Sharedlet

The figure above shows a screenshot of Sensor Sharedlet. On the top left hand corner shows a list of Sharedlets which are different applications developed under the Impromptu framework. Among them, “Robot Demo” is the Sensor Sharedlet. The user can click on the tab of the Sharedlet to open it.

Impromptu is meeting-based. There always exists a host and some participants. All participants in the meeting share the same view. Any of them can perform different actions and changes will be reflected on the screen of all participants. The bottom left hand corner shows a list of participants who are currently in the meeting. They can communicate with each other using VOIP.

After Sensor Sharedlet is opened, in the left column there are two areas for sending control messages to sensors and defining filtering criteria respectively. On the right hand side is the list of all sensors available. The list dynamically changes with the status of sensors and filtering criteria.

The presentation area contains 4 panels for displaying data of sensors. The user can decide data from which sensor he/she wants to be displayed by dragging a sensor from the sensor list to one of the panels.

E.1.1 Sensor List

On the right hand side all sensors in the operational environment is shown. Each of them provides a stream of raw data. Different types of sensor are displayed in a tree hierarchy represented by different icons, with the corresponding sensor ID next to it.

User can display the data of a particular sensor by a drag-and-drop action from the sensor list to one of the panels in the presentation area. Statuses of sensors are represented by different colors. Sensors which are currently being displayed in the presentation area are highlighted in yellow. If a sensor which was previously online is disconnected, it will be highlighted in red to catch user's attention. Offline sensors can be removed from the list by applying an empty filter in the Filter Panel.

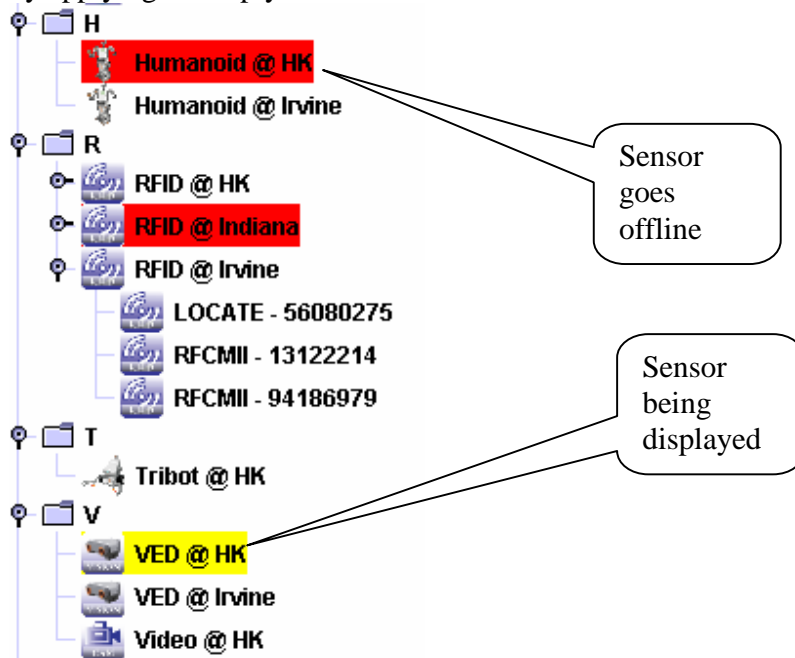
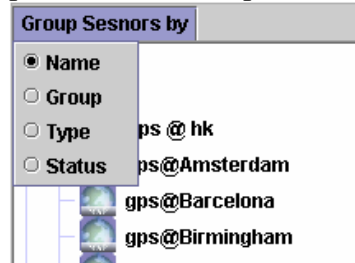


Figure 9-13 Sensors on SG sensor hierarchy highlighted in different colors

Grouping

The sensor hierarchy viewing allows user to group sensors together by different parameters. These parameters include name, group, type and status.



Group by name:



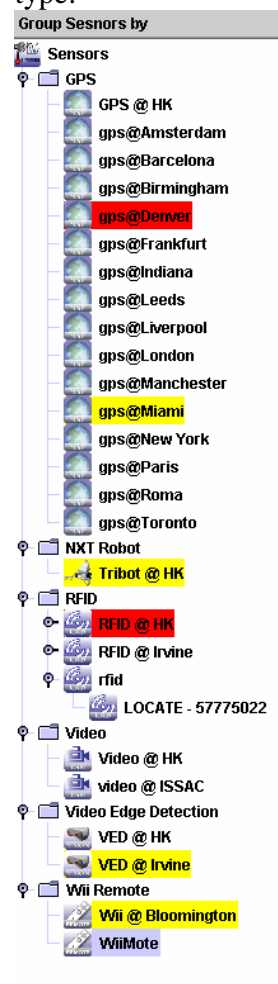
Sensors are sort by alphabetical order

Group by sensor group:



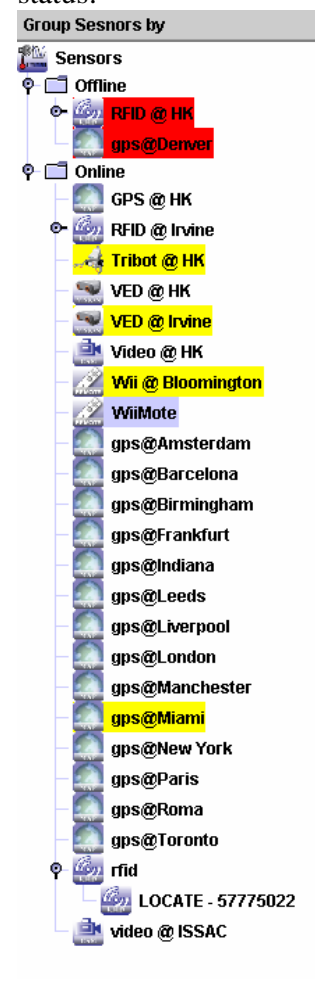
Sensors are grouped by their sensor group

Group by sensor type:



Sensors are grouped by their sensor type

Group by sensor status:



Sensors are grouped by their status

Figure 9-14 Grouping sensors into hierarchies

E.1.2 Presentation Area

The presentation area contains four panels; each of them can display data from a sensor. In order to display data of a sensor, the user has to drag and drop the icon of the sensor from the sensor list to one of the panels.

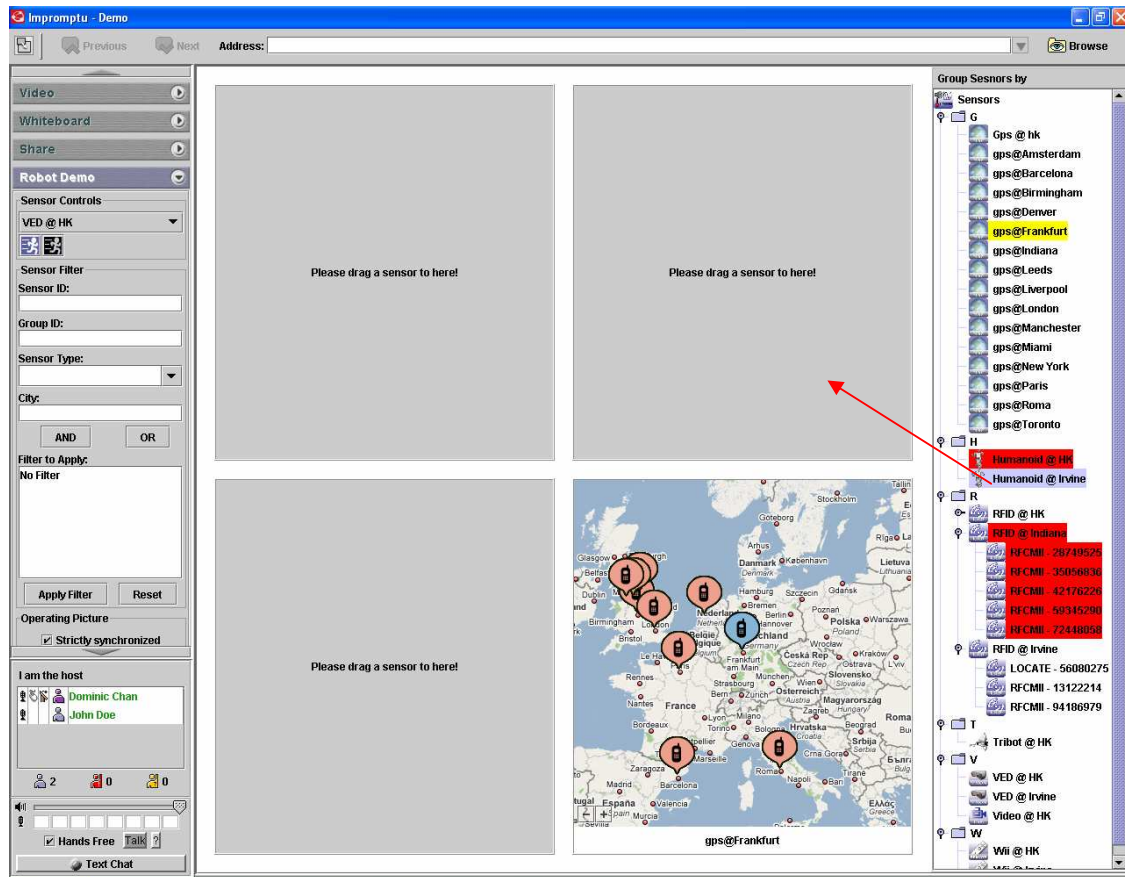


Figure 9-15 Dragging a Lego Mindstorm NXT Humanoid Robot to the top right panel

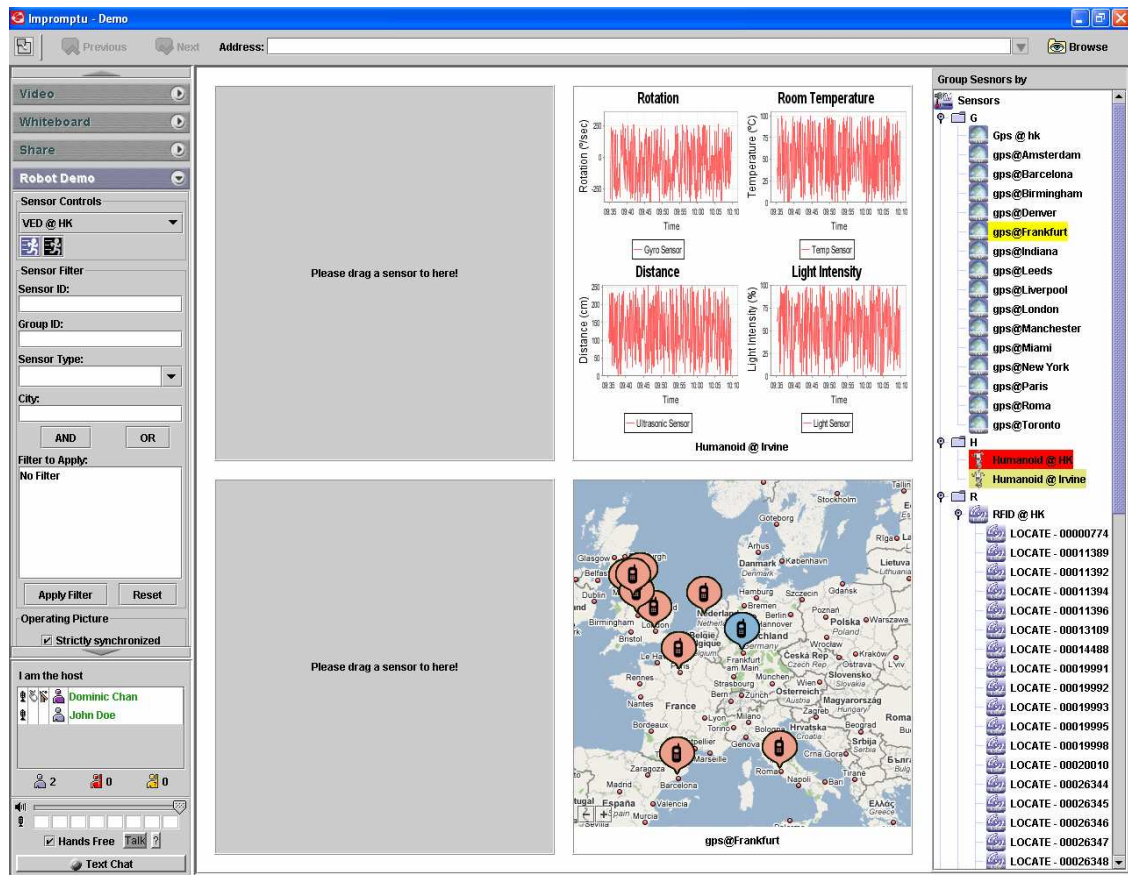


Figure 9-16 Visualization of NXT Humanoid Robot sensor streams

Each panel display can be expanded to the limit of the full presentation area by double-clicking the sensor ID label. Double-click the label again to toggle it to the original presentation panel size.

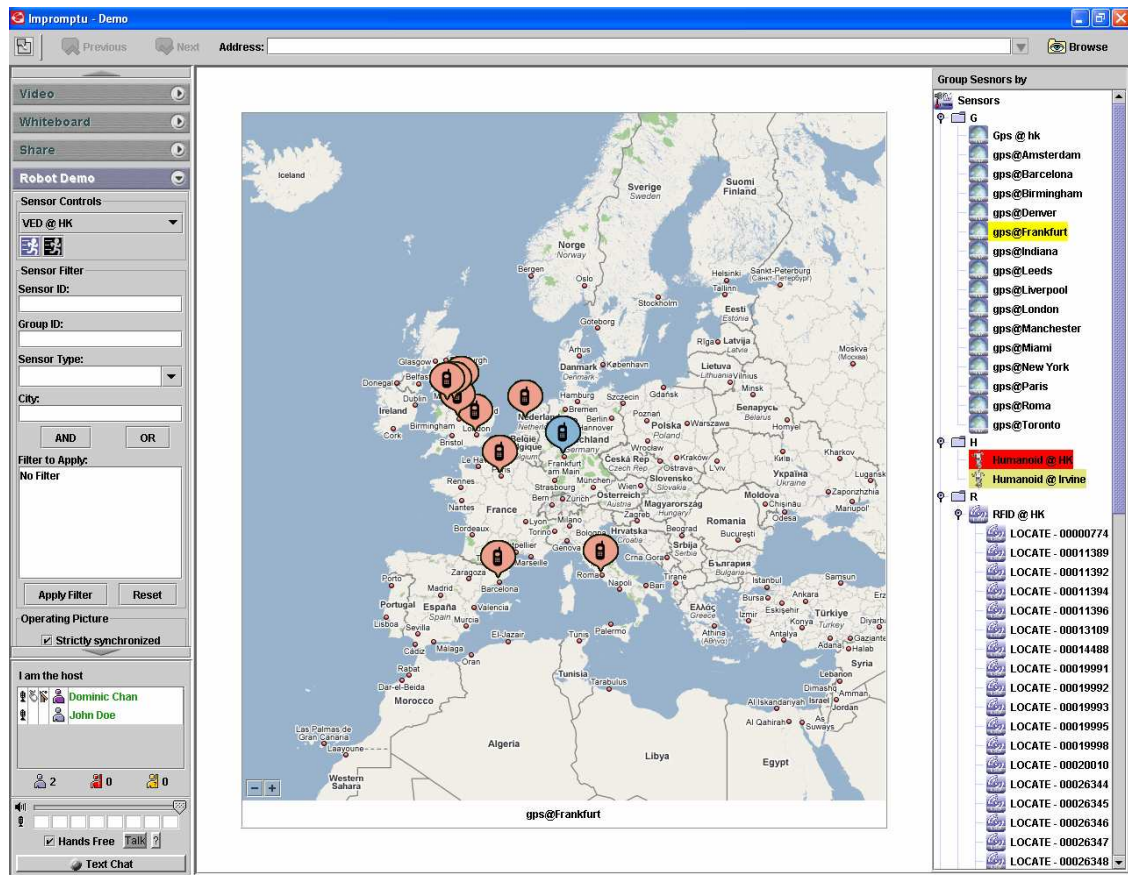


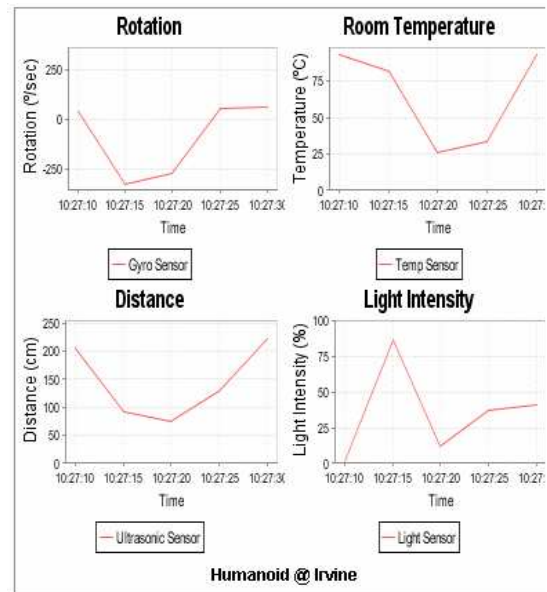
Figure 9-17 A panel displaying data from GPS device is expanded

Each type of sensor is associated with a default presentation method after being dragged to the panel. The following diagrams show the presentation used for different types of sensors.

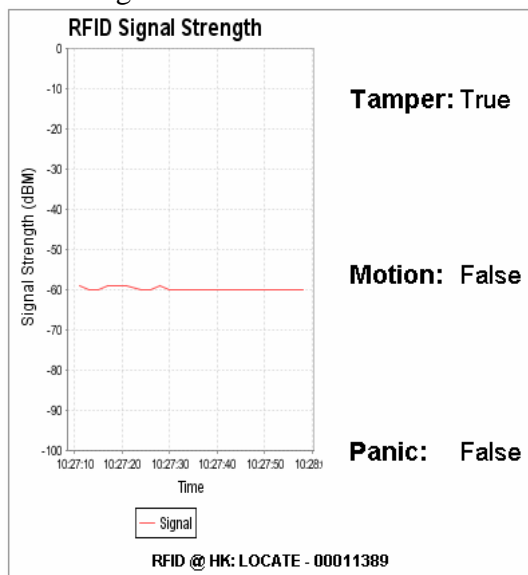
GPS



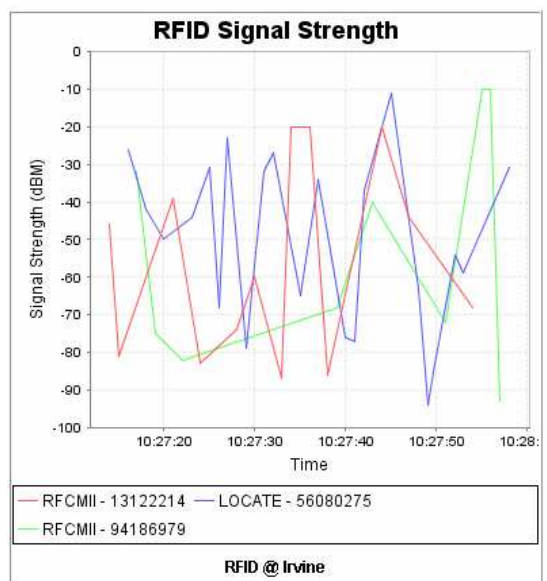
RFID Reader



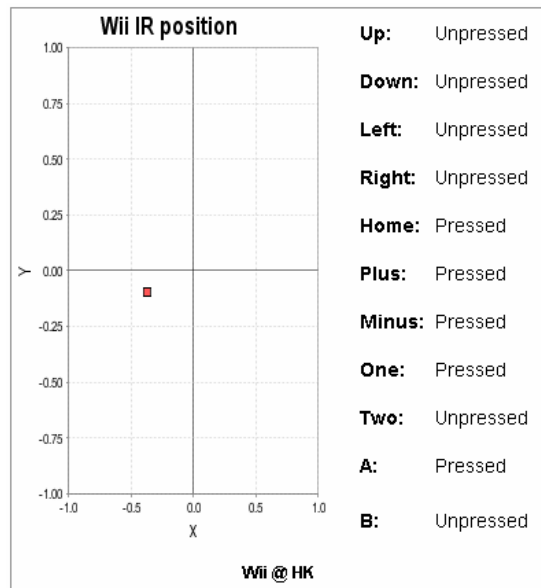
RFID Tag



NXT Robot



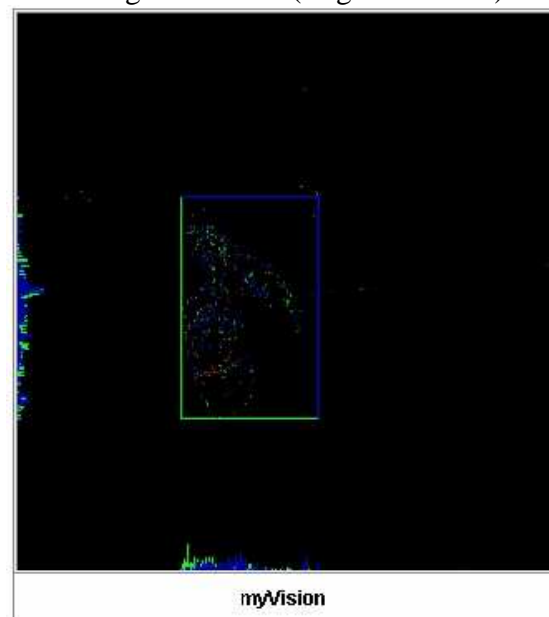
Wii Remote



Video



Video Edge Detection (Edge Detection)



Video Edge Detection (Region Finding)



E.1.3 Sending Control

To send control to a sensor, choose the sensor in the “Sensor Controls” panel. If the sensor is capable of receiving control messages, a list of buttons will be available. Click on the buttons to send the corresponding control message to the sensor.

NXT Robots can move in four directions (forward, backward, left and right) and move their arms according to the type of control messages sent. Control messages for different types of robots are slightly different.



Figure 9-18 Control panel for NXT Humanoid Robot



Figure 9-19 Control panel for XNT Tribot Robot

Another type of sensor which takes control messages are Video Edge Detection (VED) Computational Services. There are two types of control messages for VED – Edge Detection and Region Finding.

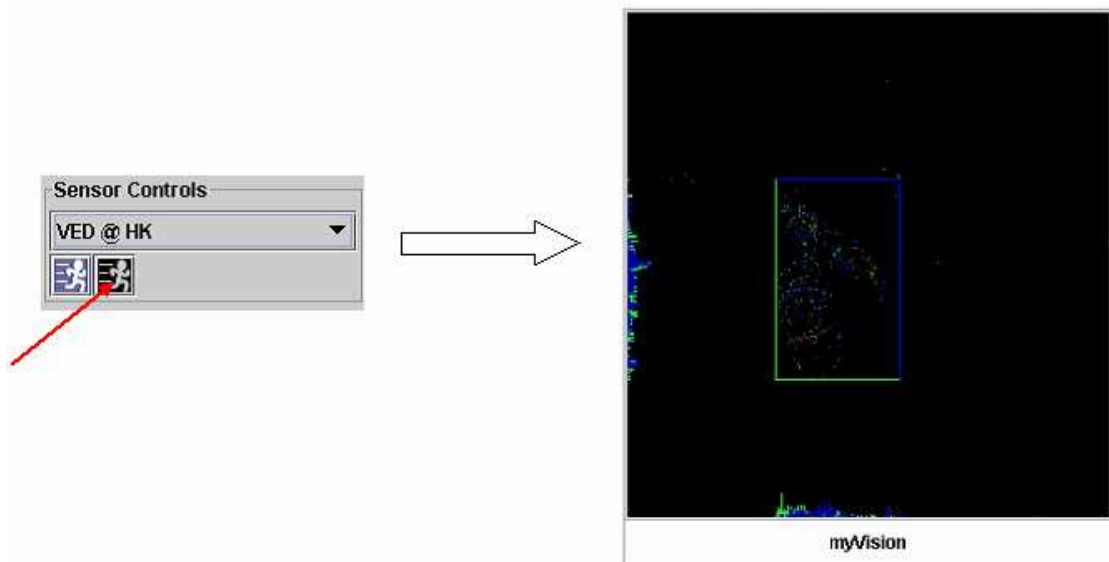


Figure 9-20 VED with Edge Detection Control

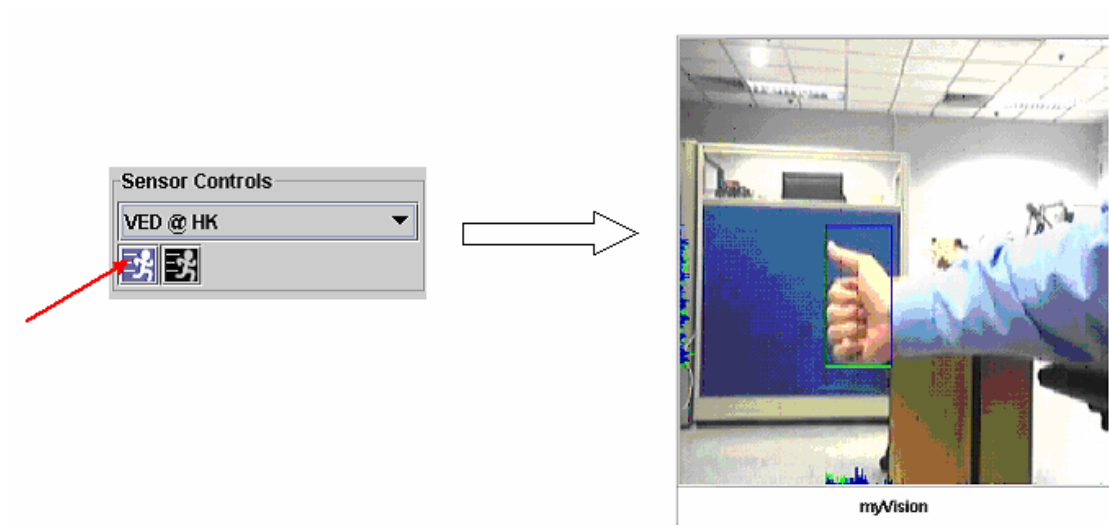


Figure 9-21 VED with Region Finding Control

E.1.4 Filtering

A user does not always want information from all sensors. He/she can filter away unwanted sensors by entering filtering criteria by using the filtering panel.

The criteria are defined by a SensorFilter object. A SensorFilter is composed of a set of properties defined in SensorProperty connected with Boolean “and” or “or” operators. Given that a list of sensor properties in a sensor filter are connected together with the “and” operator, only sensors which have properties with exact match in string comparison with ALL the properties defined in the filter should get through. Similarly sensors which have properties with 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.

To use the filtering UI, follow these steps to construct a query consists of “and” and “or” operators:

Sensor Filter

Sensor ID:

Group ID:

Sensor Type:
GPS

City:

Filter to Apply:
No Filter

Sensor Filter

Sensor ID:

Group ID:

Sensor Type:

City:

Filter to Apply:
sensorType=GPS

1. In ONE of the fields, entering the keyword which you would like to filter

2. Either press the “OR” or “AND” button to add the keyword to the list according to what your query is.

Sensor Filter

Sensor ID:

Group ID:

Sensor Type:

City:

Filter to Apply:
sensorType=GPS
[AND] groupId=hk
[OR] sensorType=RFID
[AND] groupId=group2

3. Suppose we want to show all GPS devices in group “hk” and all RFID devices in group “group2”, repeat step 1 and 2 to construct a query like this. Then click “Apply Filter”

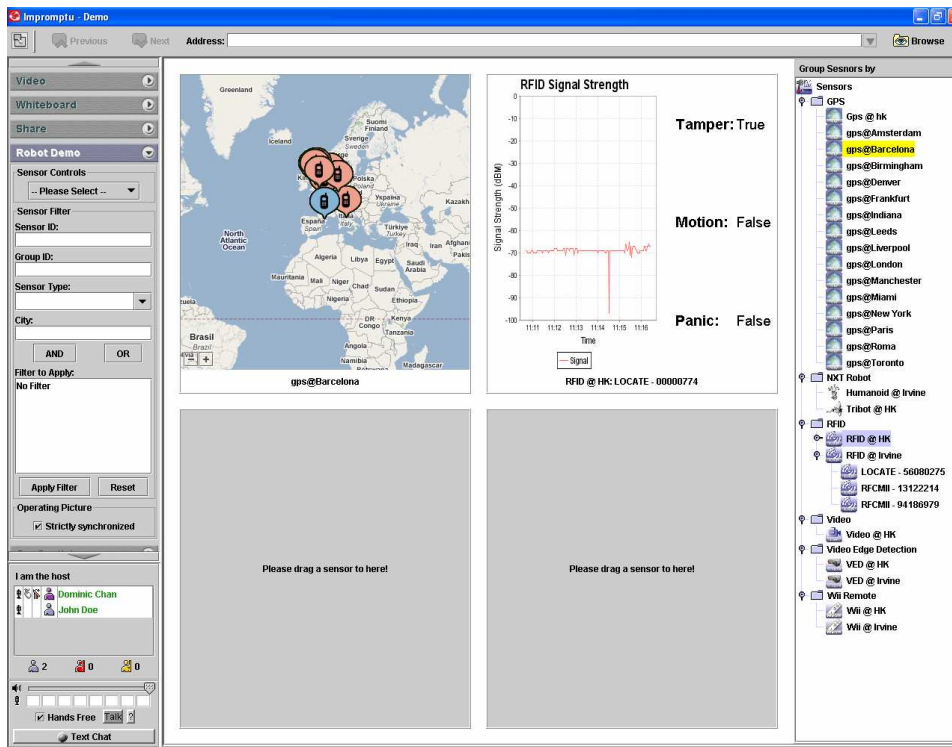


Figure 9-22 Sensor List before Filtering

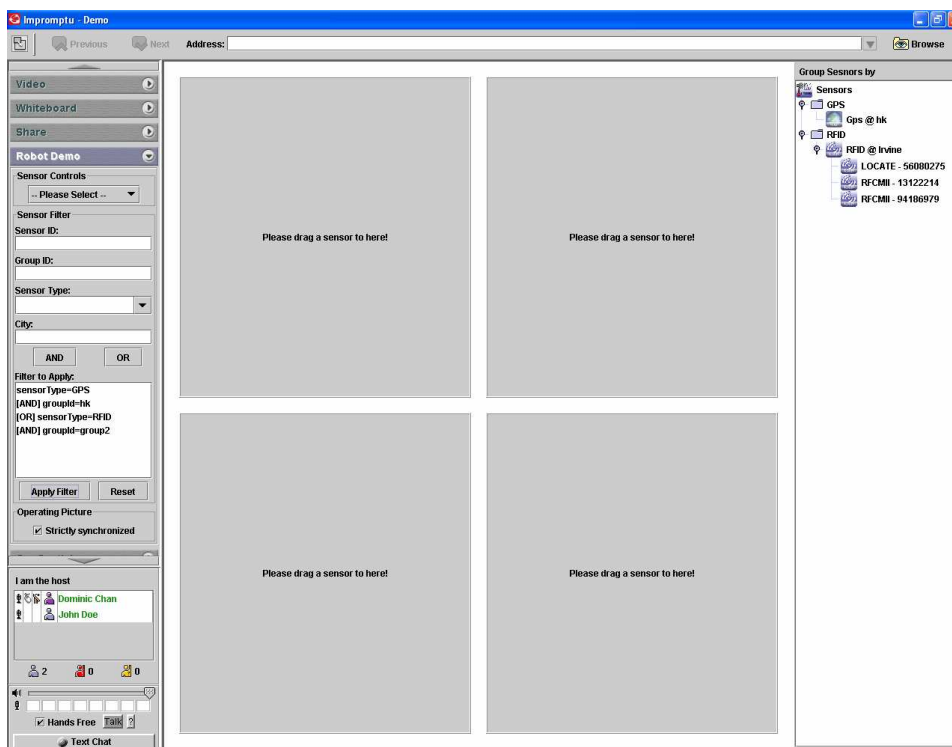


Figure 9-23 Sensor List after Filtering

E.1.5 Computational Services

Some sensors, known as “Computational Services”, do not take input from the environment. Instead, they take output of other sensors as their input, perform various computations on the data, and output the processed data finally. What computation is being performed depends on the type of service the sensor provides.

One of the Computational Services we currently have is the “Video Edge Detection (VED)”. It provides two types of services on video processing – Edge Detection and Region Finding. Edge detection is a service which detects the edges of moving objects in the video stream. Detected edges are visualized as colorful lines out of the black background. Region Finding is similar to Edge Detection but uses the original video as background. Both services are encapsulated in a single Video Edge Detection. Two different algorithms are requested by sending two different control messages.

To set which video stream as the source of a VED, you can drag the icon of a video on the sensor list to a VED icon. Afterwards, drag the video service to one of the display panels from right to left. The processed video should be shown.

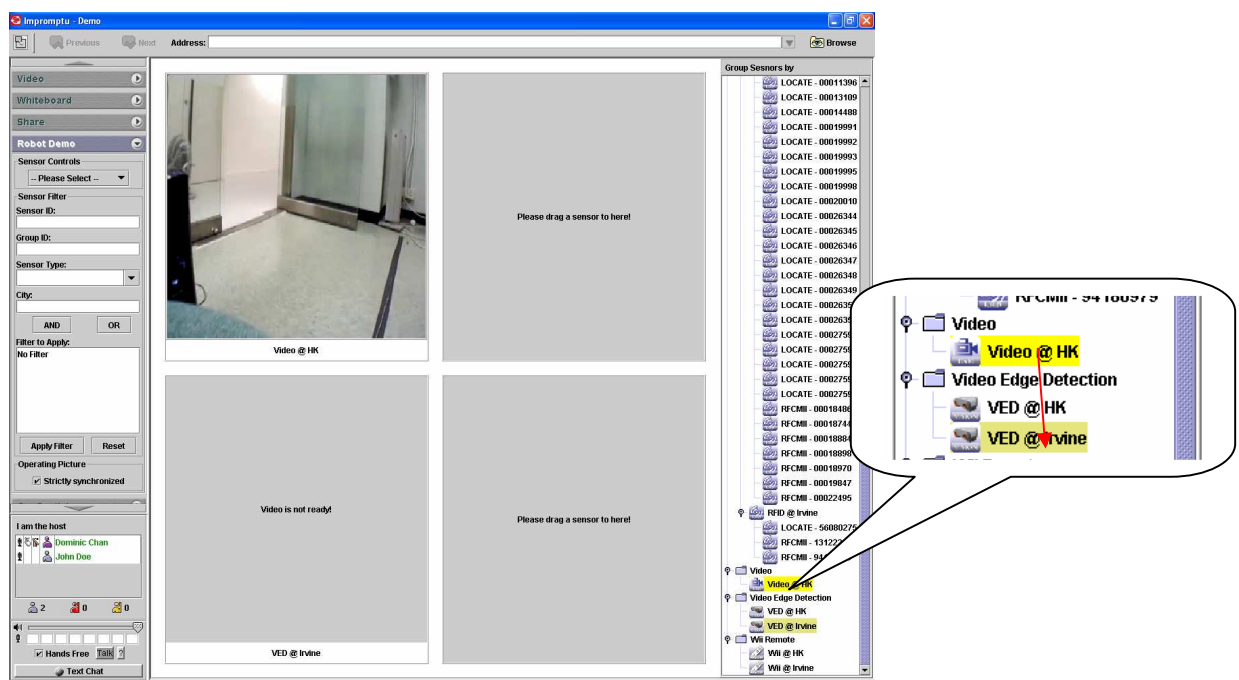


Figure 9-24 Setting the Source of Video Service

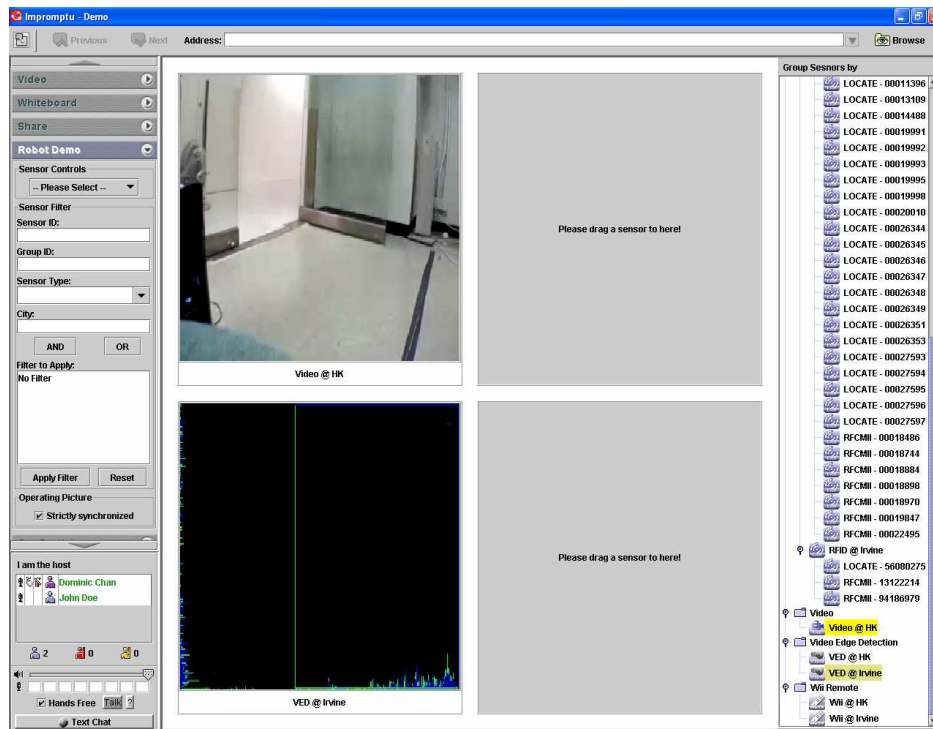
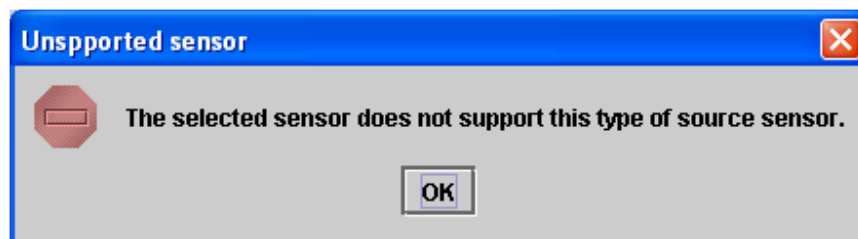


Figure 9-25 Source of VED has been set

Fault tolerance issue: If any attempt to drag a sensor to a Computational Service which does not support that particular sensor type, a warning message will be displayed. For example, an attempt to drag a GPS sensor to a Video Computational Service results in the following error message:



E.1.6 Synchronization Mode

To support UDOP to the fullest extent, Sensor Sharedlet provides two synchronization modes among meeting participants – Strictly Synchronous and Loosely Synchronous. Strictly Synchronous means that all participants are sharing the same operating picture, with sensors being displayed in all panels in the presentation area and filtering criteria being the same. Every attempt to change the operating picture by any of the participants will be reflected on the screen of all participants.

On the other hand, Loosely Synchronous means every participant has his/her own operating picture. The user can choose to display data from any of the sensors in the

presentation area and define his/her filtering criteria without affecting the view of other participants.

Only the host of the meeting has the right to switch between Strictly and Loosely Synchronous modes.

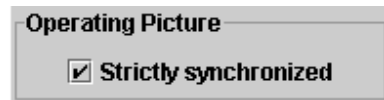


Figure 9-26 Panel for setting mode of operating picture

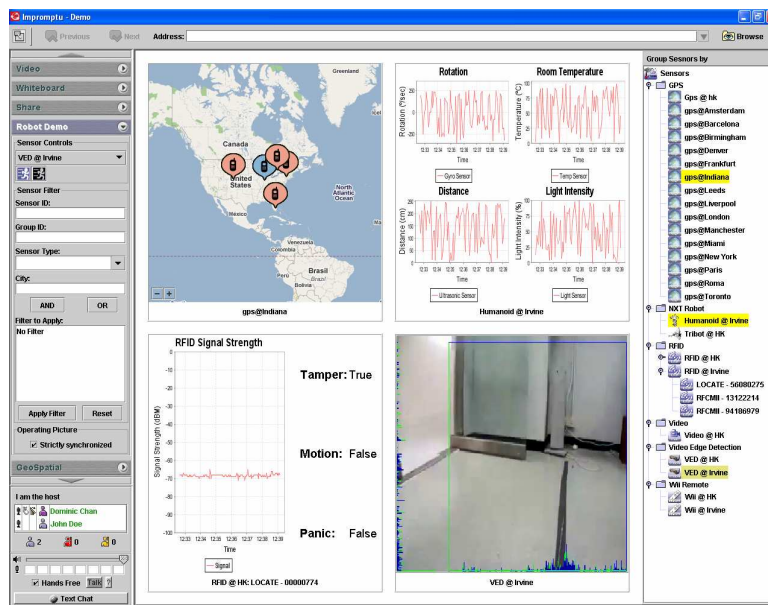


Figure 9-27 View of meeting host in Strictly Synchronous Mode

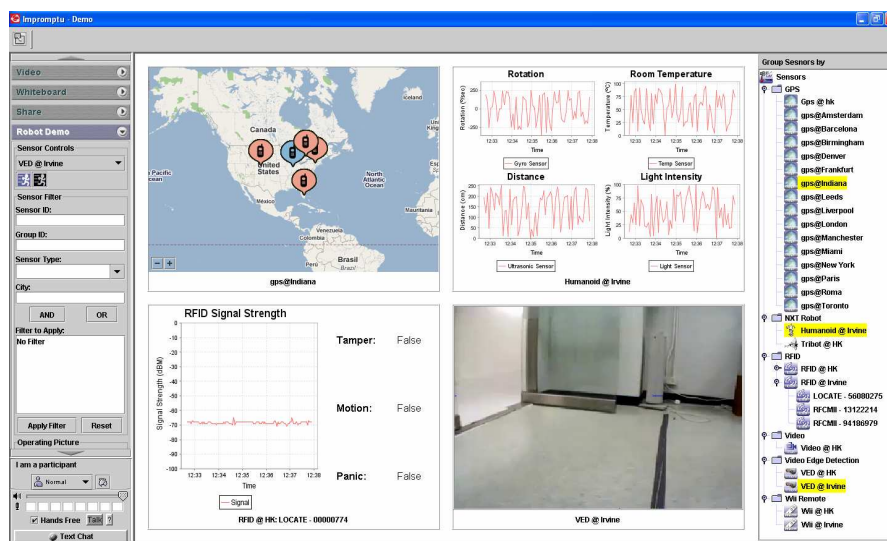


Figure 9-28 View of meeting participant in Strictly Synchronous Mode



Figure 9-29 View of meeting host in Loosely Synchronous mode

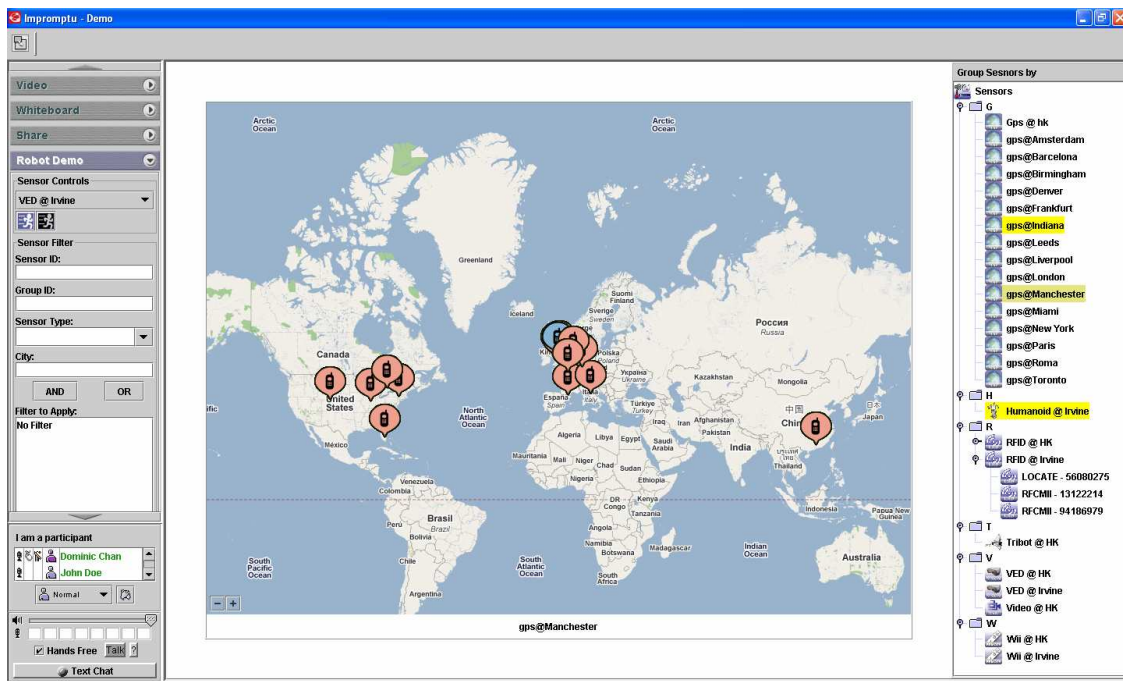


Figure 9-30 View of meeting participant in Loosely Synchronous mode

E.2 Geo-Spatial Sharedlet

Geo-Spatial Sharedlet is another Sharedlet which was developed under Impromptu. The purpose of this Sharedlet is to provide an operating picture for displaying the geo-spatial location of all sensors being deployed through SCGMMS with a 2D world map representation. Every sensor is represented by a numbered icon in the map with its sensor ID displayed on the right hand column respectively.

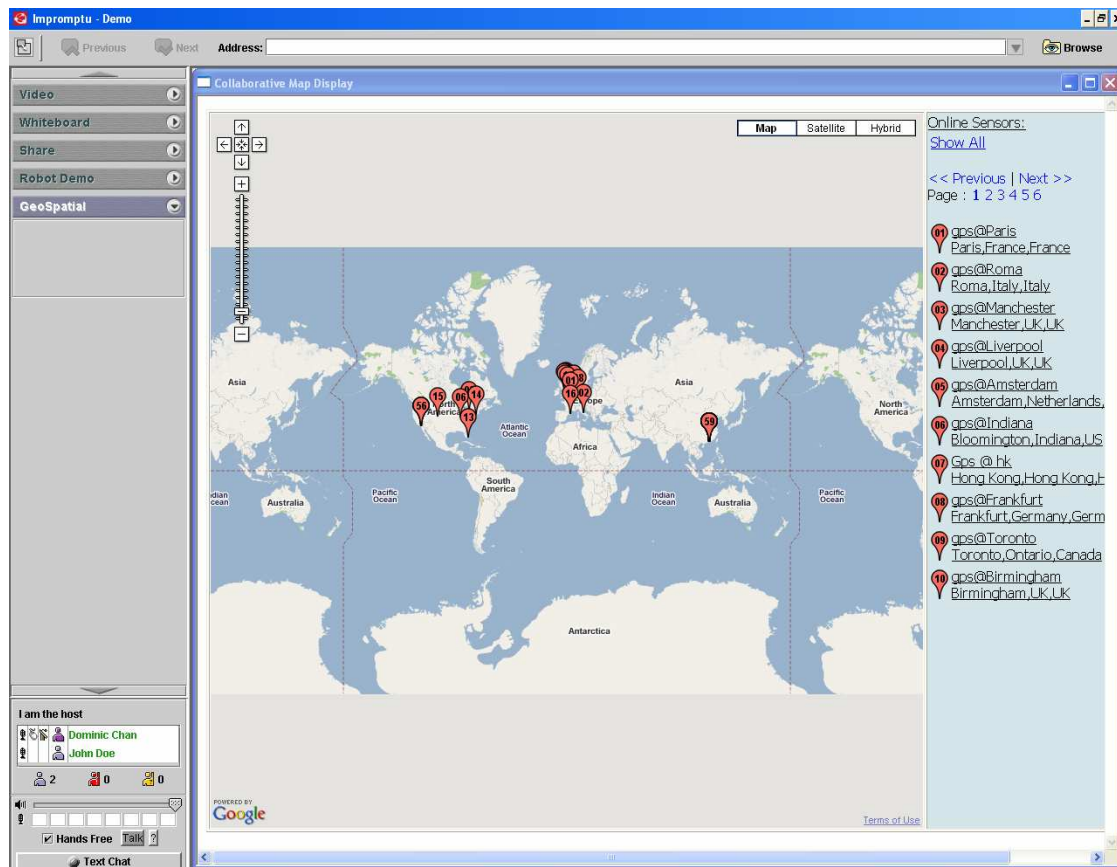


Figure 9-31 The Geo-Spatial Sharedlet

The location of a sensor is determined by the following way:

If the sensor is a GPS device, the location of the sensor is determined by the data streamed out of the device. For other types of sensors, their location are determined by the street address entered during deployment (the address field is mandatory). The sharedlet automatically translates the address to a 2D lat-lng position by using Google Maps API [4].

To view the information of a particular sensor, simply click on the icon.

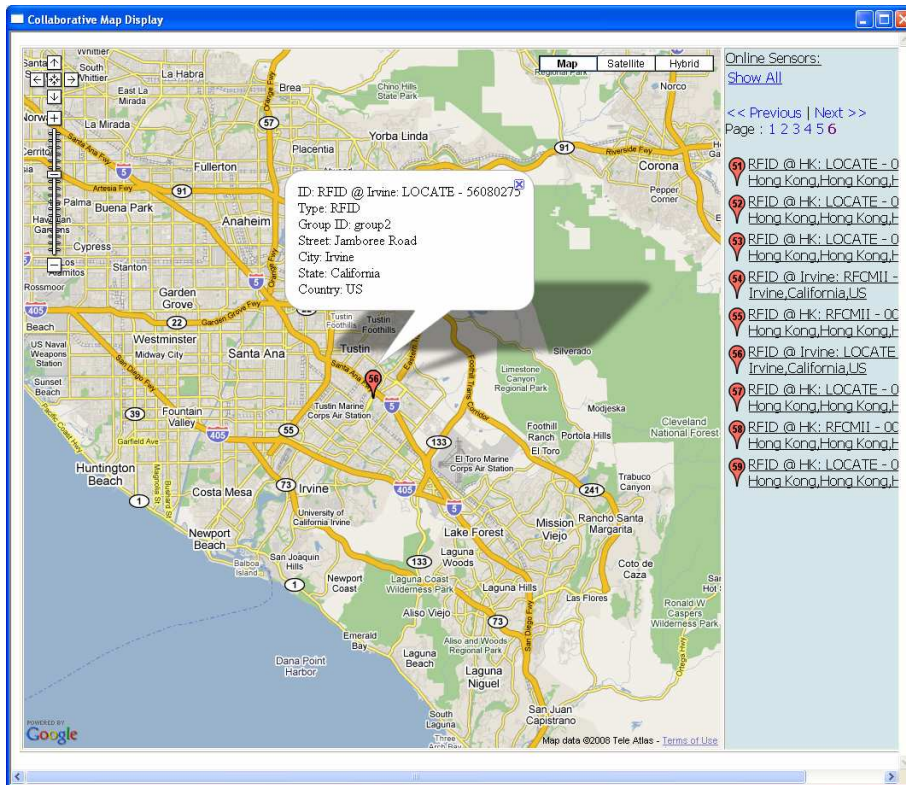


Figure 9-32 Information of a sensor

A strictly synchronous view is shared among all meeting participants.

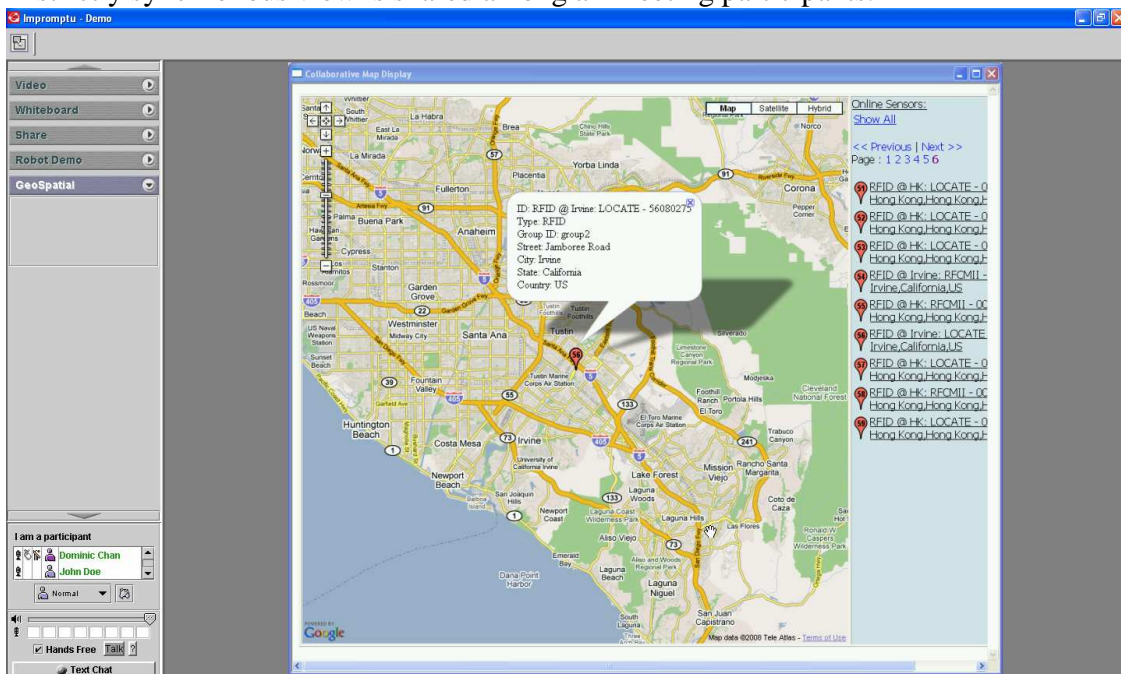


Figure 9-33 View of meeting participants

E.2.1 User-interface for UDOP Applications

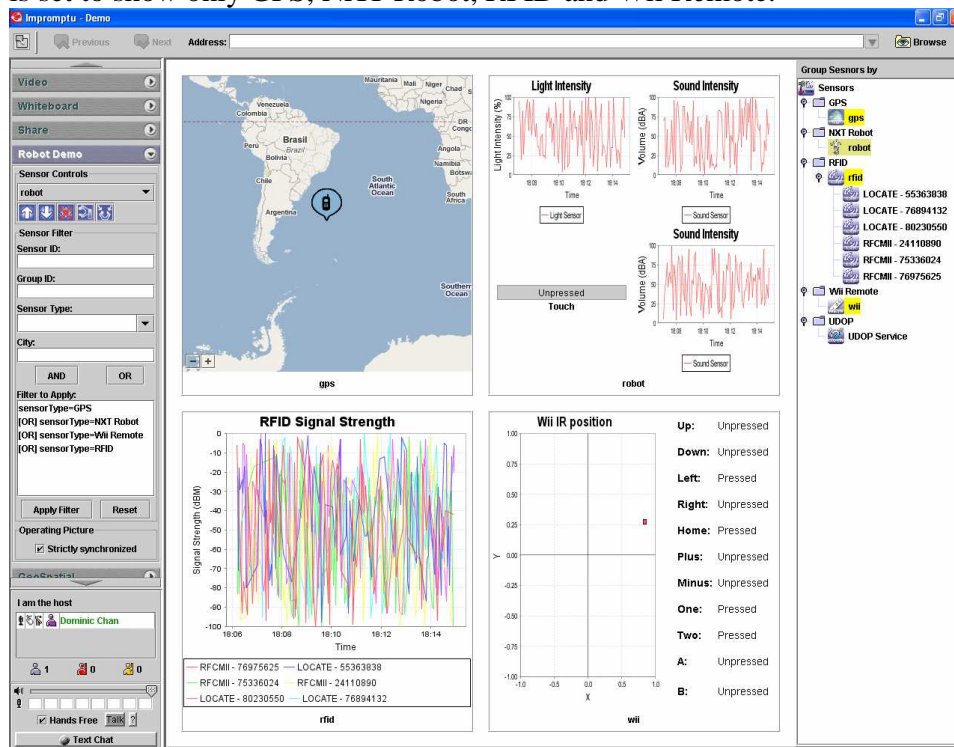
Having architectural support from SCGMMS, this section will illustrate how a front-end application supports UDOP Management.

Creating the Operating Picture

In Sensor Sharedlet, you can define the operating picture in 3 perspectives:

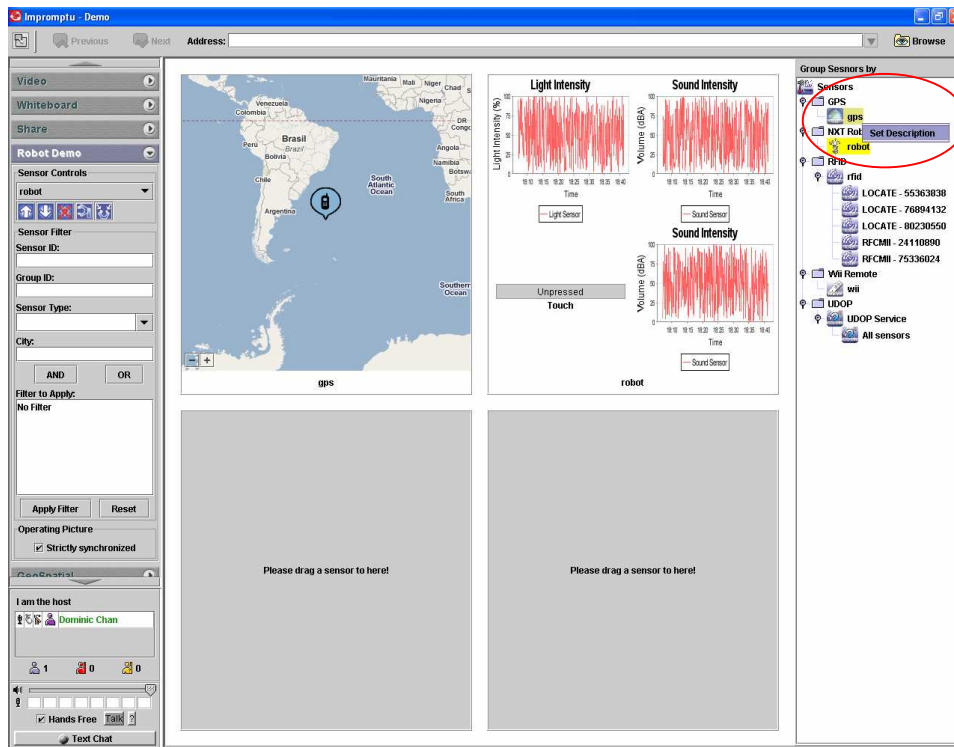
1. Showing the data of which sensor in the presentation area by drag-and-drop?
2. What filtering criteria are defined?
3. What descriptions given to some of the sensors?

The figure below shows a sample operating picture showing data of 4 sensors. The filter is set to show only GPS, NXT Robot, RFID and Wii Remote.

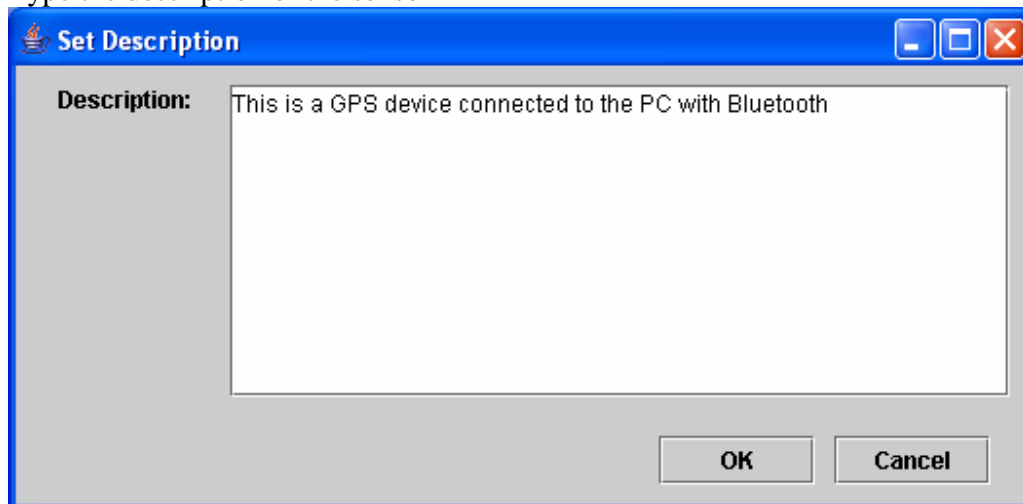


To set the description of a sensor, follow these steps:

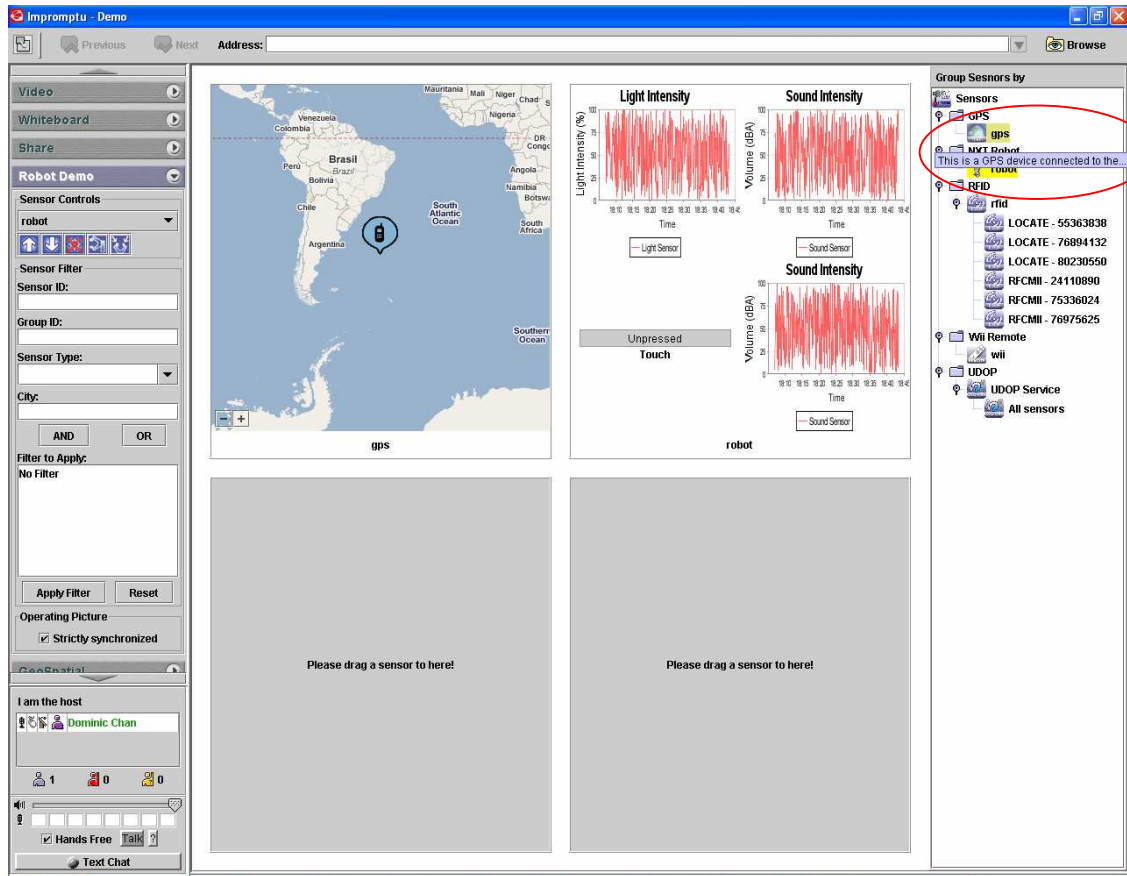
1. Right-click on the sensor and choose “Set Description”



2. Type the description of the sensor



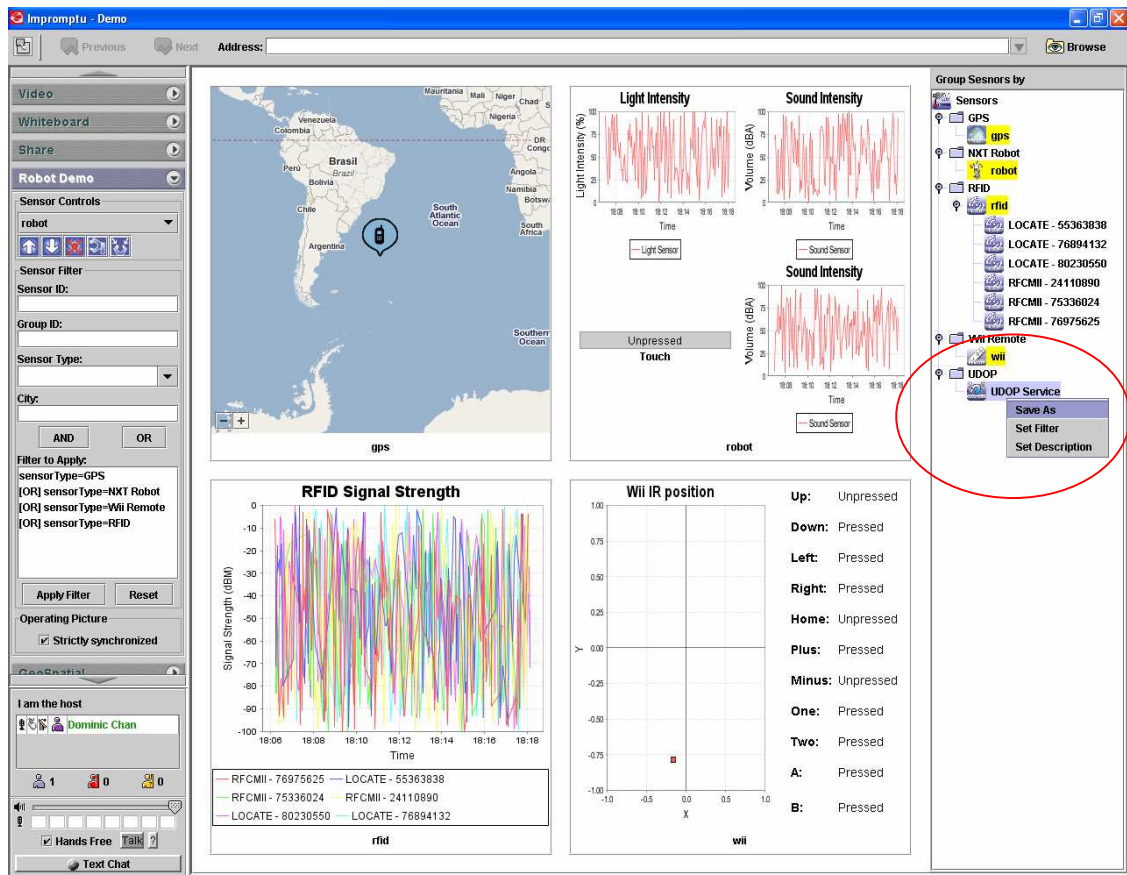
3. The description can be viewed as tool-tip text when the mouse is over the icon of the sensor in the sensor list



Saving a UDOP Template

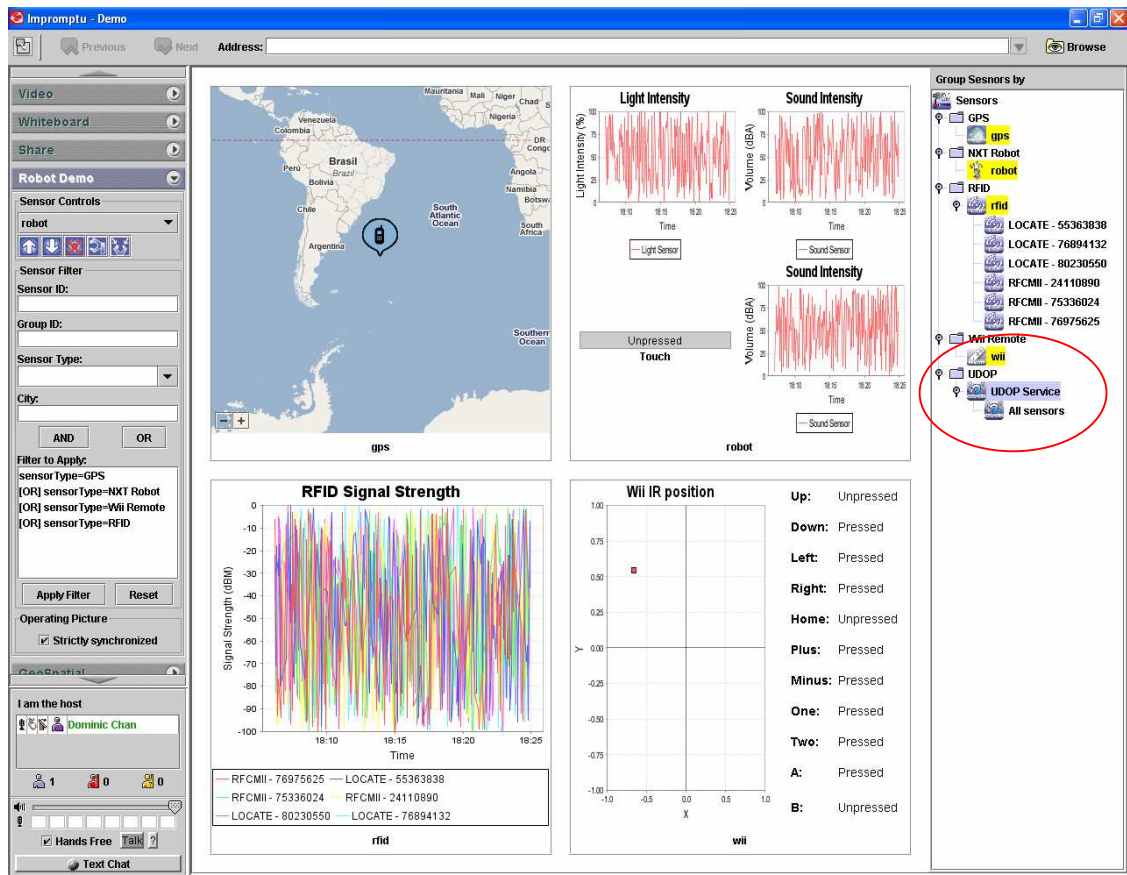
Once the operating picture is defined, the user can save it to a UDOP Template by using the UDOP Service. To save, follow these steps:

1. Define your operating picture, then right-click “Save As” on “UDOP Service” in the sensor list.



2. Enter the Label and Description of the UDOP Template and press “ok”

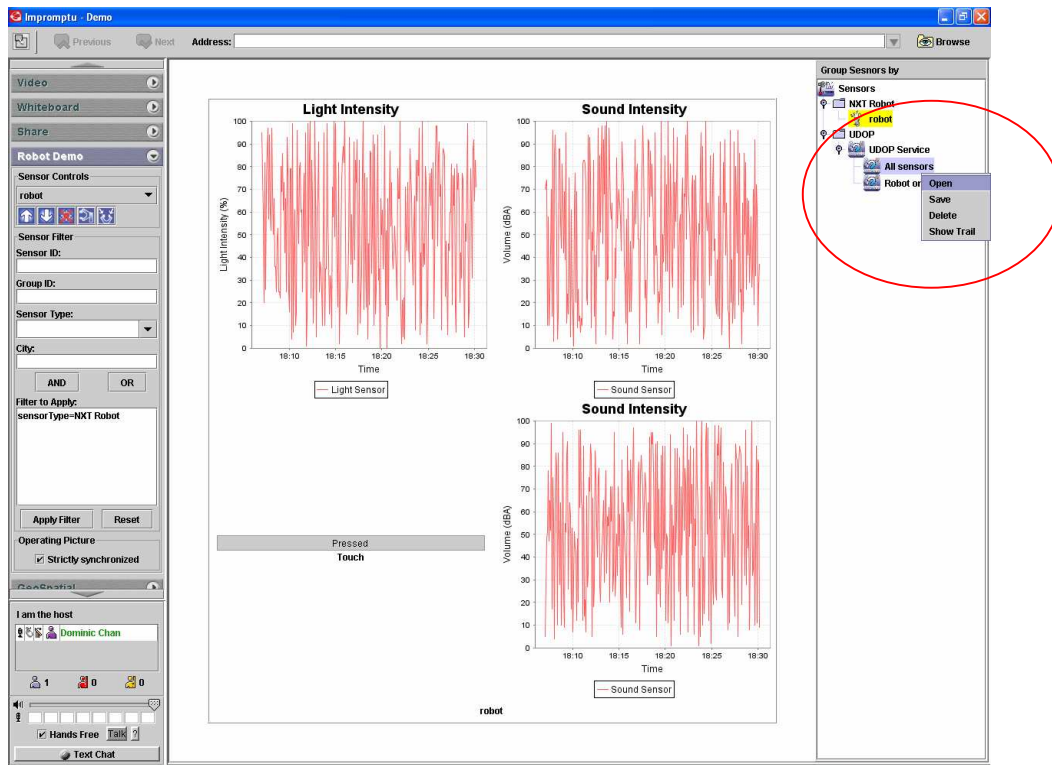
3. You can see that the new Template has been added to the list.



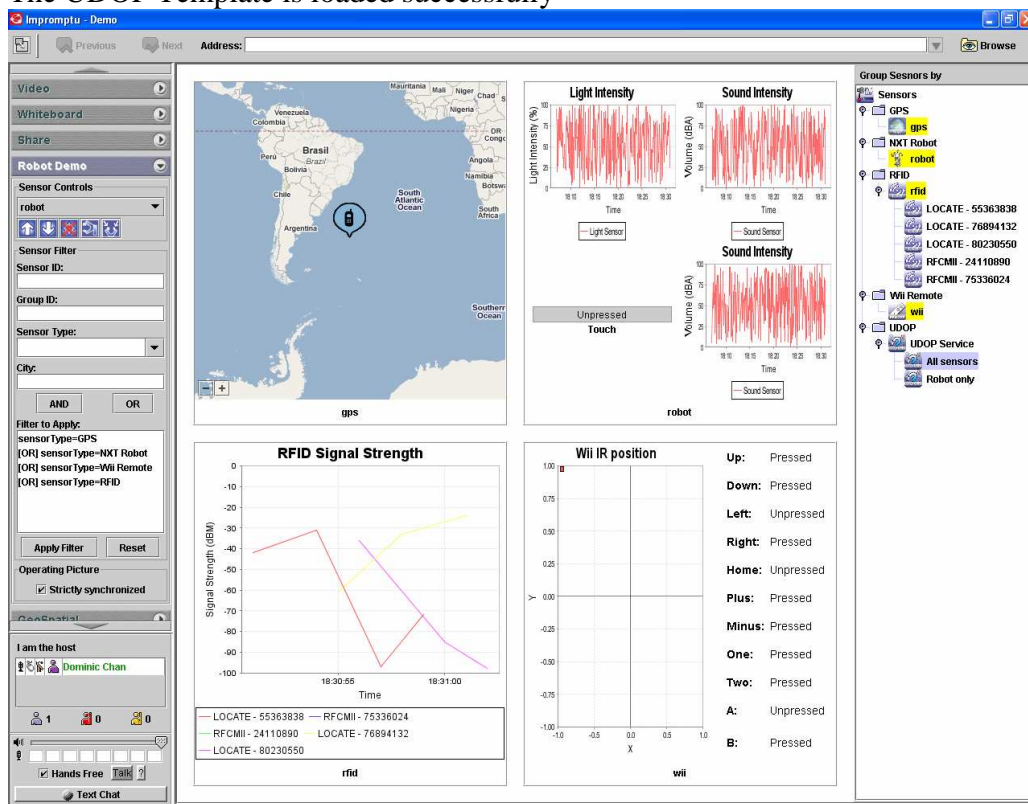
Opening a UDOP Template

You can load UDOP Templates from UDOP Service by following these steps:

1. Supposed the current operating picture is showing the view of a single NXT Robot and the user wants to load the Template “all sensors”. Right-click on the Template and press “Open”

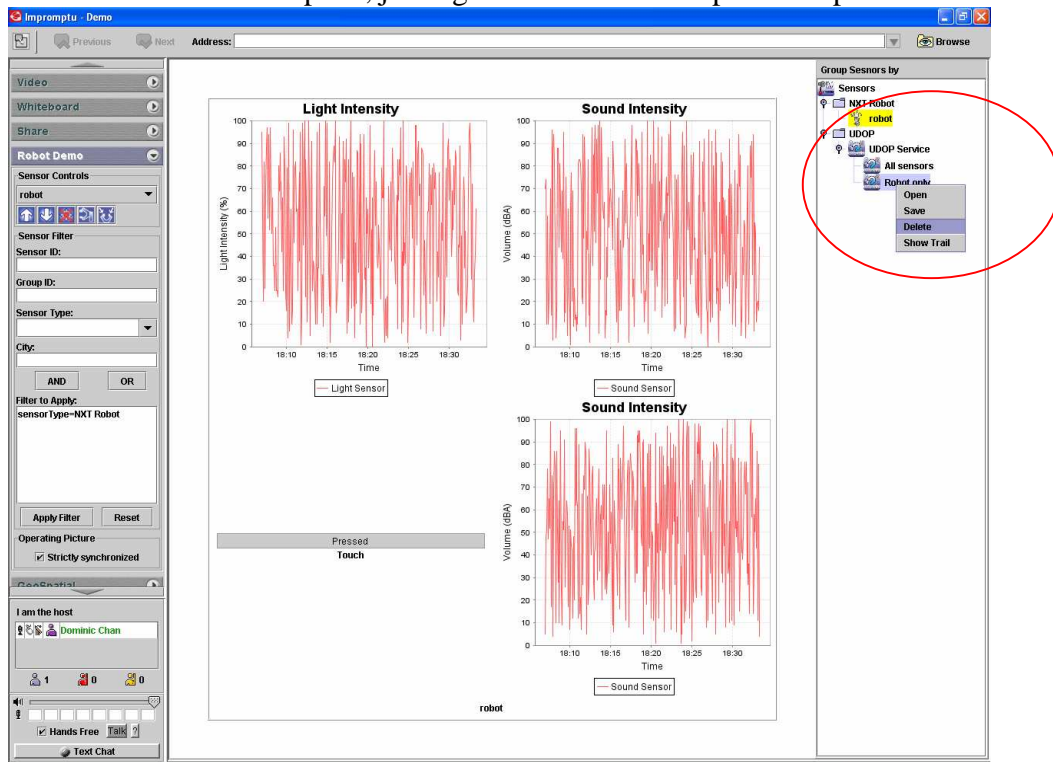


2. The UDOP Template is loaded successfully

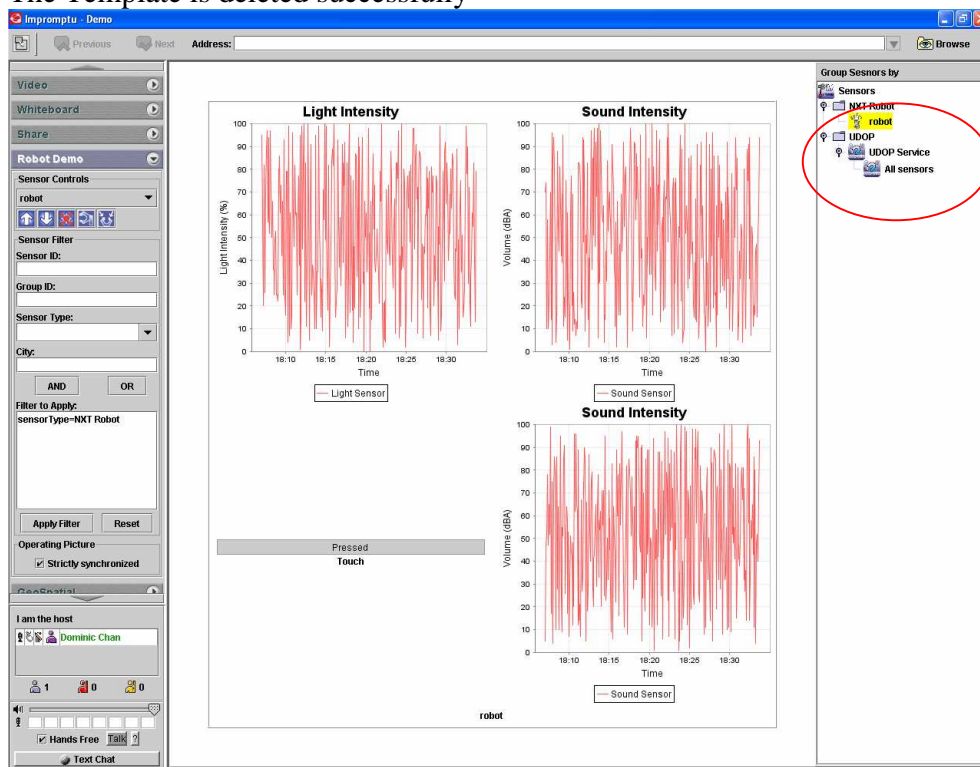


Deleting a UDOP Template

1. To delete a UDOP Template, just right-click on the Template and press “delete”



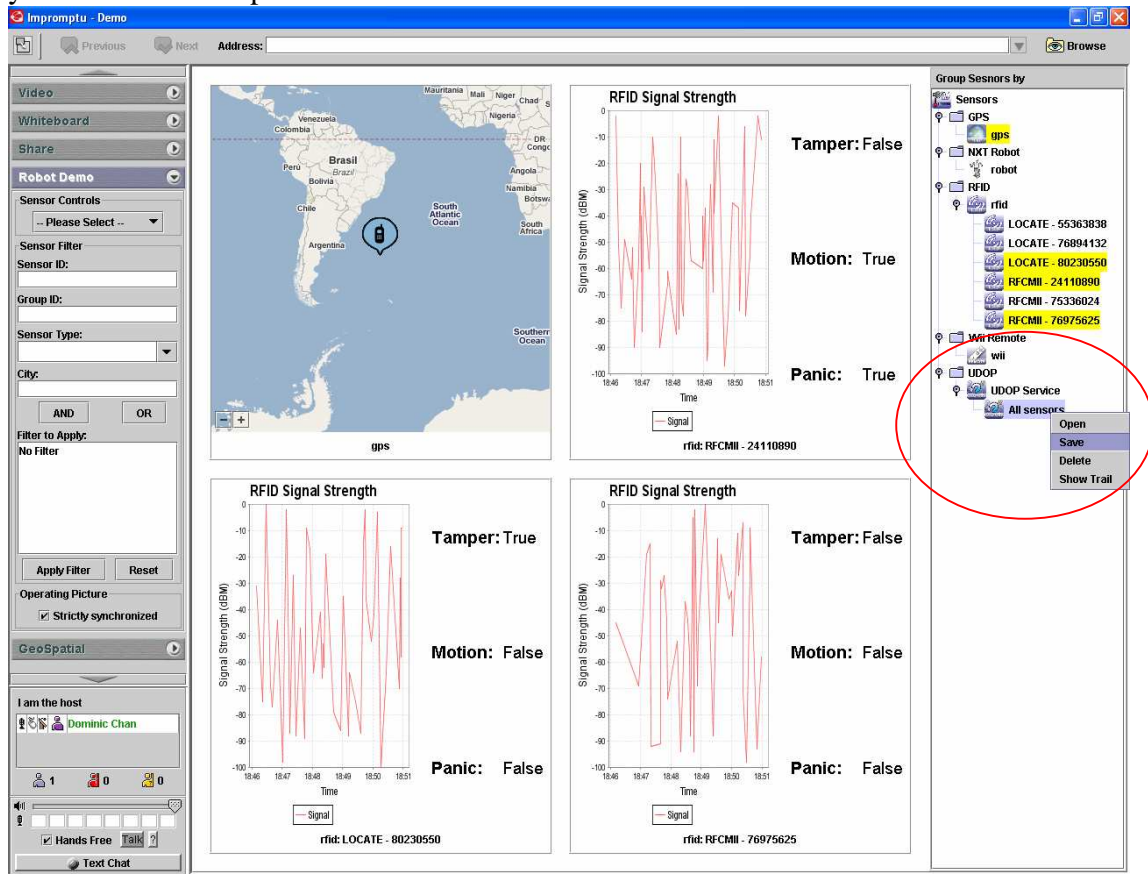
2. The Template is deleted successfully



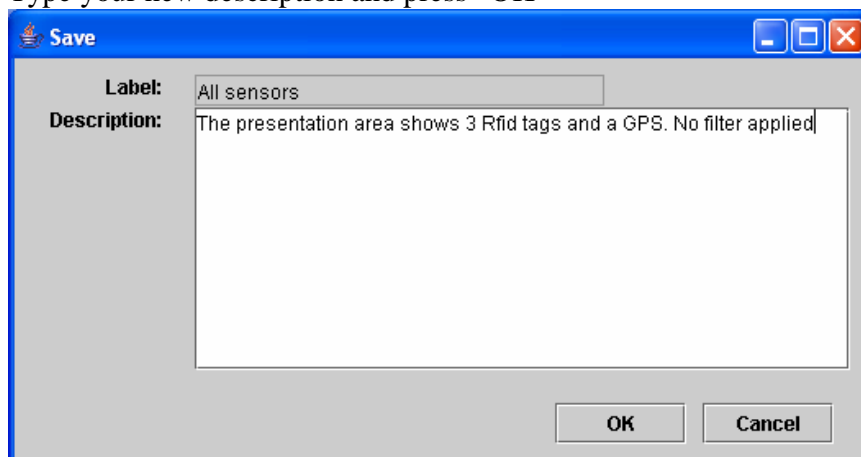
Showing Change History of a UDOP Template

The user can update a UDOP Template by using the “save” function. Each subsequent updated will be recoded as a growing list and they can be loaded. Please follow these steps to learn how to update, browse and load change history of a UDOP Template.

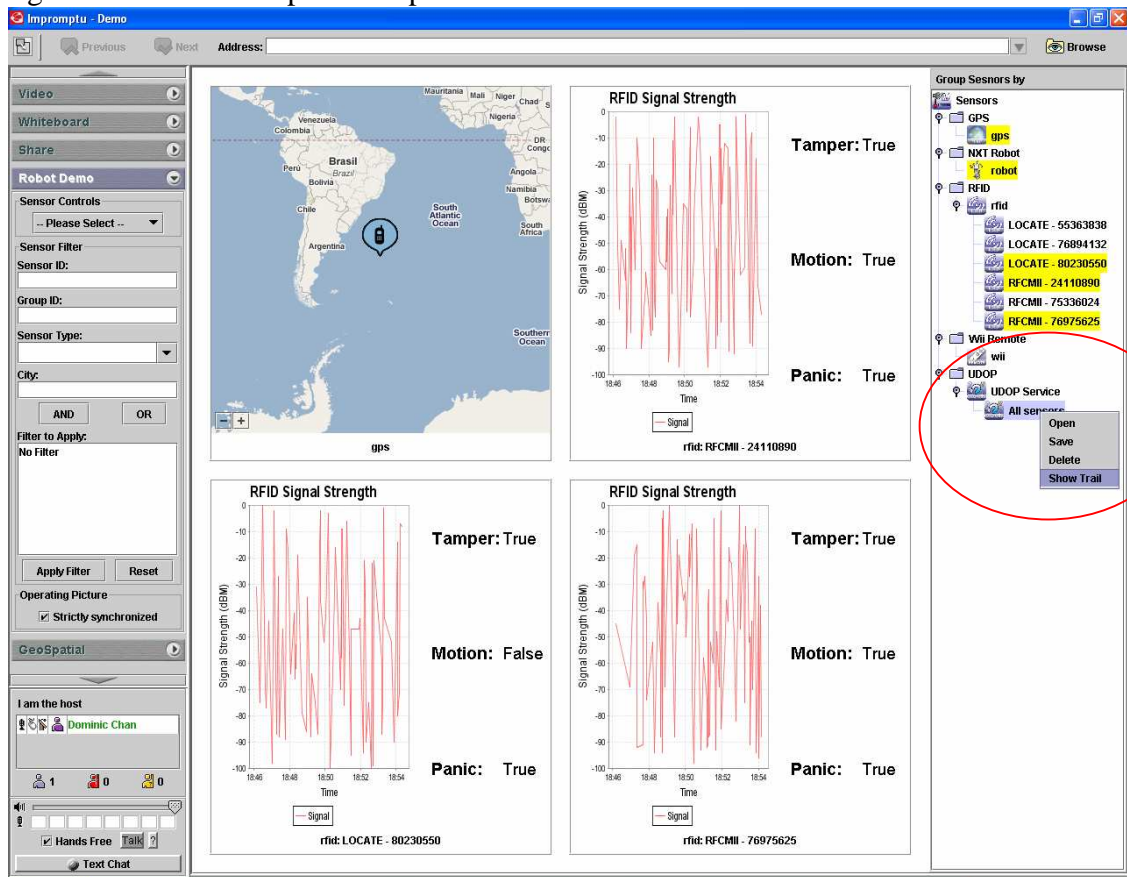
1. Once you have defined your operating picture, right-click on the UDOP Template you would like to update and click “save”



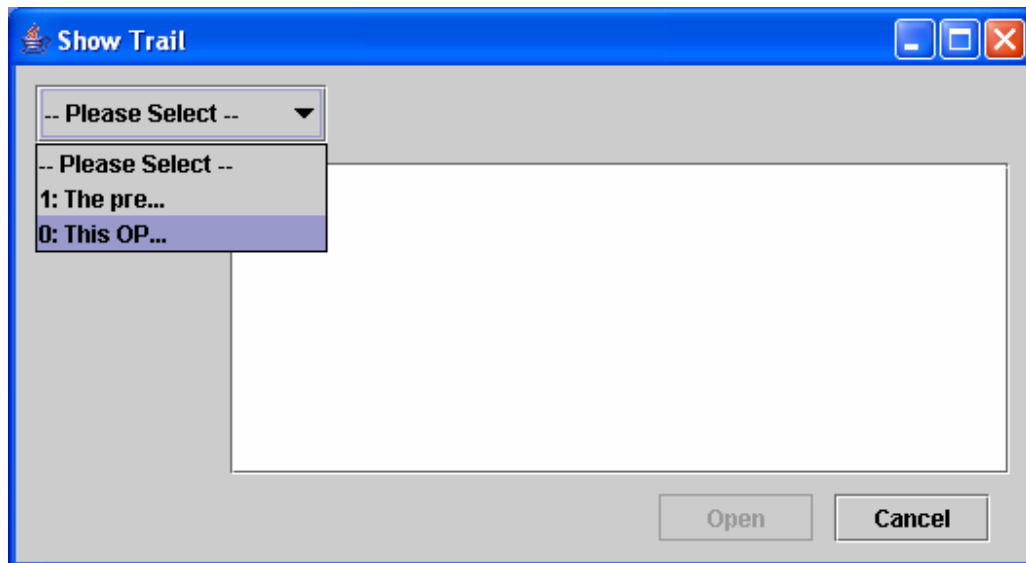
2. Type your new description and press “OK”



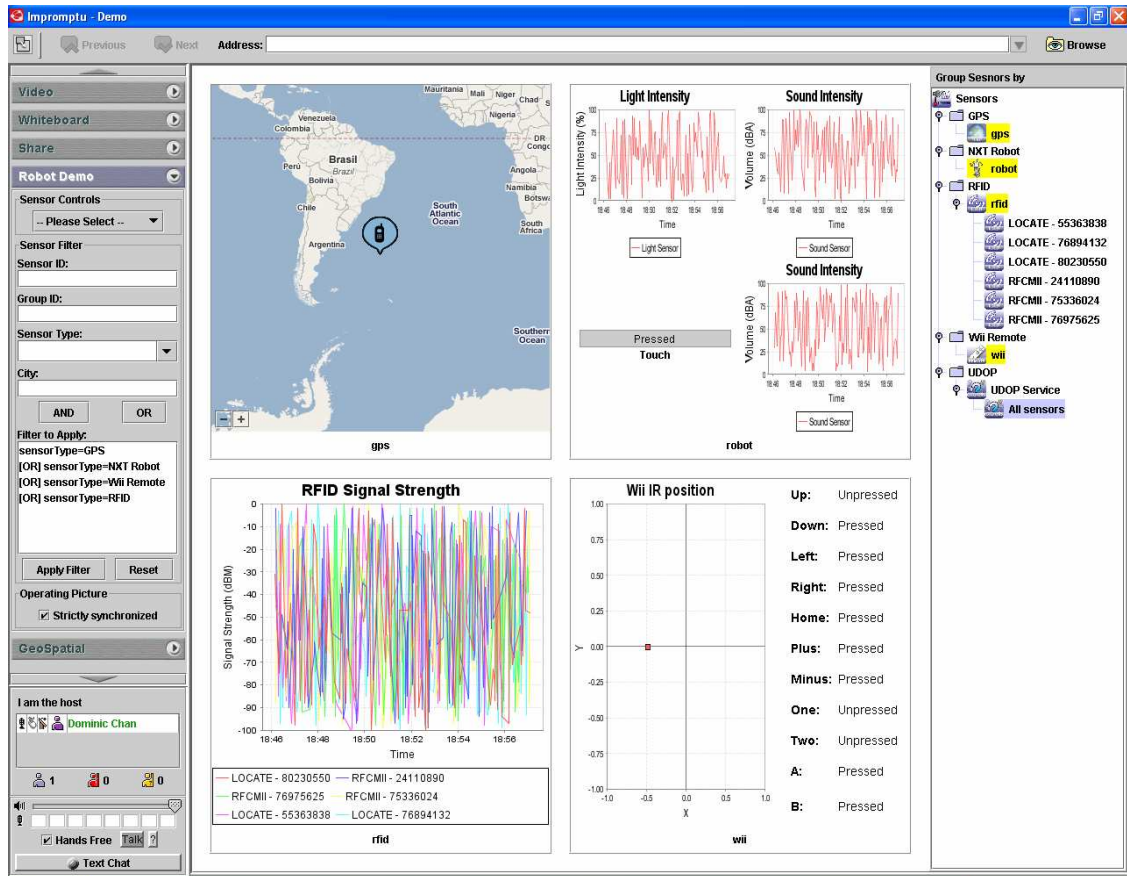
- The Template is updated. You can browse the update history of the Template by right-click on the Template and press “Show Trail”



- So far we have updated this Template once. The previous entry can be found at the bottom of the list



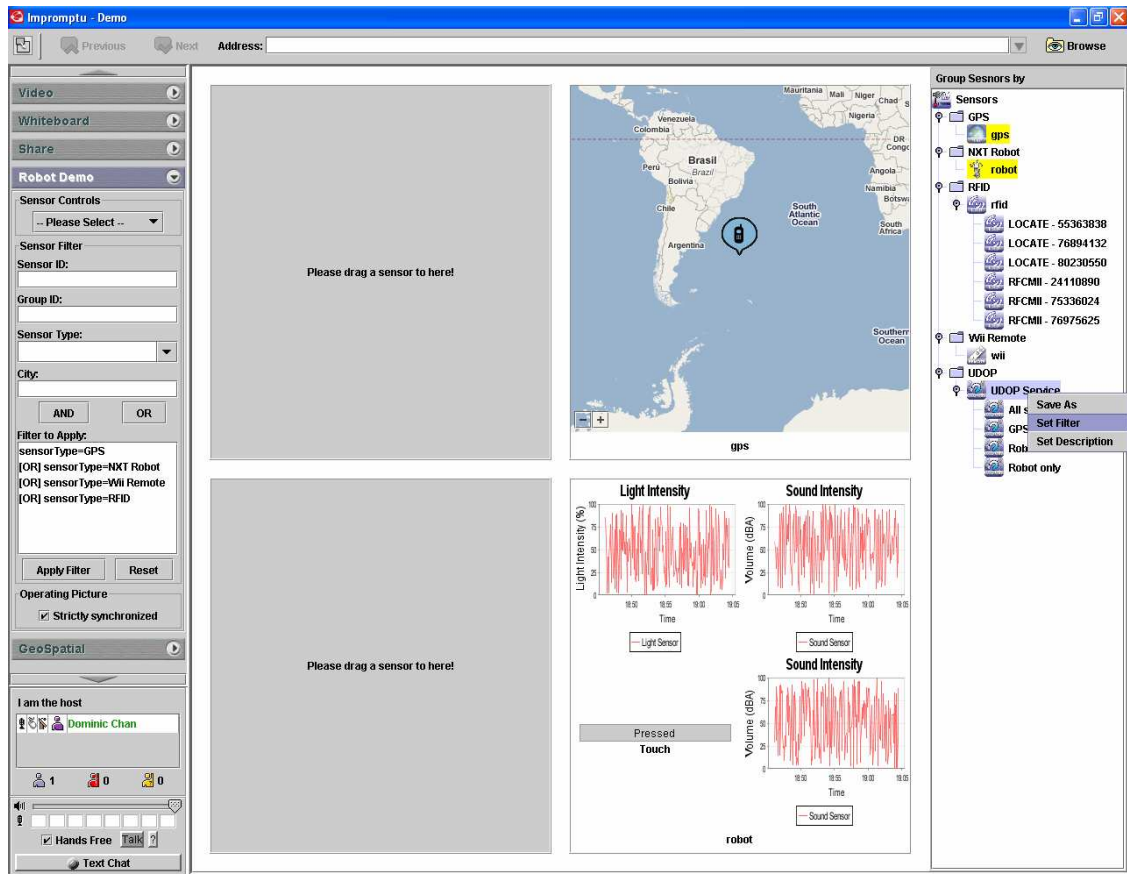
- To open it, choose it in the combo box and press “Open”. The previous Template is loaded.



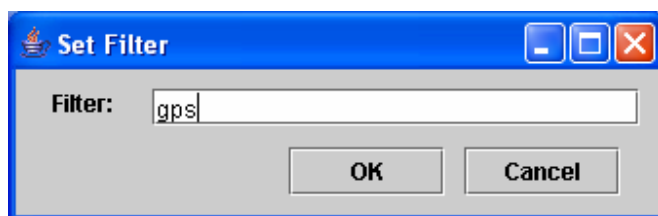
Filtering UDOP Templates

The user can filter UDOP Templates by using keywords. The system will search through the labels and descriptions of all UDOP Templates and show only those which match with the keyword. To do filtering, follow these steps:

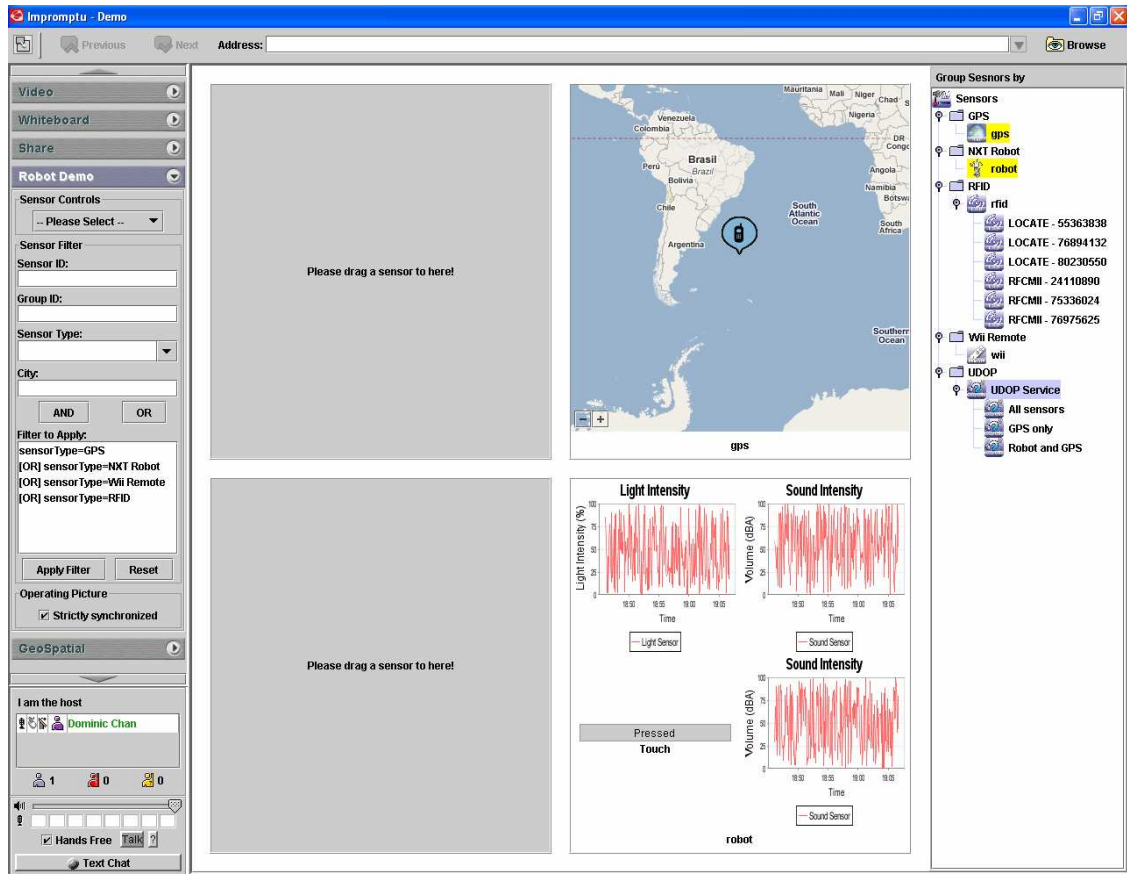
1. Right-click on UDOP Service and choose “Set Filter”.



2. Enter the keyword and press “OK”. Here we would like to find all Templates with “gps”.



3. Only 3 Templates left now



Appendix F - RFID Positioning (Localization)

RFID reader: RFCode M220 Reader

RFID Tag: RFCode M100 Active Tag

Introduction

There are many studies about RFID positioning in both academic and commercial world. In commercial world, one of the most successful products is AeroScout System. They use both Time Difference of Arrival (TDOA) and Received Signal Strength Indication (RSSI) algorithm to determine tags' location with 3 of its dedicated AeroScout Location Receivers. (For details, please refer to <http://www.aeroscout.com>). The study by Applied Physics Institute, Western Kentucky University showed that its outdoor accuracy was less than 1m but its indoor accuracy was as large as 6m [RFIDWomble].

In academic world, there are 2 algorithms on focus. One is LANDMARC (LocAtion iDentification based on dynaMic Active Rfid Calibration)[RFIDNi]. Another one is LEMT (*Location Estimation using Model Trees*)[RFIDYin][RFIDYin2]. [RFIDNi] said that LANDMARC's indoor accuracy was: 50 % of errors were within 1 meter while the maximum error distance was around 2 meters with 4 RFID readers. [RFIDYin][RFIDYin2] said that LEMT's indoor accuracy was: 40% of errors were within 0.5meter and 80% of errors were within 1.5 meters with a considerable number of readers and reference tags.

According to our initial testing, with the following algorithm, errors were around 0.5 meter with one reader and one tag only.

Model Building Algorithm:

The signal strength (power per unit square) received by a RFID reader from a RFID tag is inversely proportional to the square of the distance between the reader and the tag. So we have:

$$P \propto \frac{1}{r^2}$$

However, the output of the signal strength received by a RFID reader is in dBm. To express an arbitrary power P as x dBm, we have:

$$P = 10^{\frac{x}{10}}$$

Assume that the signal strength received by a RDID reader from a RFID tag depends on their distance and their surrounding disturbance according to a multiplicative model as follow:

$$P \propto \frac{1}{r^2} \cdot (\text{Environmental Factor})$$

Suppose that there are a reference tag and a target tag, which follow multiplicative models as below respectively:

$$P_{T \text{ arg et}} \propto \frac{1}{r_{T \text{ arg et}}^2} \cdot (\text{Environmental Factor}_{T \text{ arg et}})$$

$$P_{\text{Re f e r e n c e}} \propto \frac{1}{r_{\text{Re f e r e n c e}}^2} \cdot (\text{Environmental Factor}_{\text{Re f e r e n c e}})$$

Further assume that the effect of the environmental factors on the 2 tags is similar, then we can cancel the environmental factors in the way as follows:

$$\frac{P_{T \text{ arg et}}}{P_{\text{Re f e r e n c e}}} \propto \frac{r_{\text{Re f e r e n c e}}^2}{r_{T \text{ arg et}}^2}$$

By taking logarithm, we can obtain a linear model as follows:

$$\ln P_{T \text{ arg et}} = a_0 + a_1 \ln P_{\text{Re f e r e n c e}} + a_2 \ln r_{T \text{ arg et}} + a_3 \ln r_{\text{Re f e r e n c e}} + \varepsilon$$

By rearranging, we have:

$$\ln r_{T \text{ arg et}} = a'_0 + a'_1 \ln r_{\text{Re f e r e n c e}} + a'_2 \ln P_{T \text{ arg et}} + a'_3 \ln P_{\text{Re f e r e n c e}} + \varepsilon'$$

Substituting $P = 10^{\frac{x}{10}}$, we have:

$$\ln r_{T \text{ arg et}} = b_0 + b_1 \ln r_{\text{Re f e r e n c e}} + b_2 x_{T \text{ arg et}} + b_3 x_{\text{Re f e r e n c e}} + \varepsilon''$$

This is a linear model on which our positioning program is based.

Position Detection

Having built a model from a number of training data, one could test the model by setting up known positions and apply the position detected by the model for comparison.

For an initial special case, the model will be applied to detect a position collinear with and between the 2 RFID readers. In order to reduce the fluctuation of the detected

position due to the instability of signal strengths detected, the system has implemented some smoothing technique:

1. Weighted average of the distances from 2 readers

Suppose there are 2 readers: Reader 1 at a position p_1 and Reader 2 at a position p_2 with $p_1 < p_2$. Let s_1 and s_2 be the distance of a target tag from Reader 1 and Reader 2 estimated by the detected signal strengths with the model respectively. Also, let s_{12} be the distance between the 2 readers. Then, the position of the target tag p_t is calculated in the following way:

$$p_t = (p_1 + s_1) \times \frac{\frac{1}{s_1}}{\frac{1}{s_1} + \frac{1}{s_2}} + (p_2 - s_2) \times \frac{\frac{1}{s_2}}{\frac{1}{s_1} + \frac{1}{s_2}}$$

2. Moving average

After p_t is obtained by the weighted average as above, the estimated position is further smoothed by the moving average technique. Let $p_{t,N(T)}$ be the position of the target tag p_t estimated at time T. Then the k-step moving average at time T is:

$$\bar{p}_{t,N(T)} = \frac{p_{t,N(T)} + p_{t,N(T)-1} + p_{t,N(T)-2} + \dots + p_{t,N(T)-k+1}}{k}$$

$N(T)$ is the number of the position detection recorded by the system. $T=0$ is the time when the system starts.

Lists of Acronyms, Abbreviations, and Symbols

Term	Meaning
Sensor-Centric Grid Middleware Management System (SCGMMS)	A Middleware for sensor management
Sensor	a time-dependent stream of information with a geo-spatial location
User-defined Operating Picture (UDOP)	An operating picture with all its aspect being defined by the user dynamically
Common Operating Picture (COP)	An operating picture which is common to all the participants involved in a session
Loosely synchronous model	In the system, there exists one user who is the presenter. Under certain condition (e.g. during presentation), all participants have the COP same as the presenter. Under other condition, each participant can have his/her own operating picture
Grid Builder (GB)	Part of SCGMMS for management of sensors
Sensor Grid (SG)	Part of SCGMMS for brokering sensors, applications and GB
Sensor Sharedlet	A sample UDOP application developed on Impromptu's Sharedlet framework which utilize SCGMMS
Computational Service	Sensors which does not take input from the environment. Instead, they take output of other sensors as their input, perform various computations on the data, and output the processed data finally
Sensor Service Abstraction Layer (SSAL)	A common interface for Sensors and Computational Service to communicate with SCGMMS

End of Document