

Development of the Intelligent Sensor Network Anomaly Detection System: Problems and Solutions

Leon Reznik and Carll Hoffman

Abstract— The paper describes the development of the Sensor Network Anomaly Detection System (SNADS). SNADS is designated to become a framework to support signal change detection in sensor networks (SN), the backbone application of a crucial importance for design of other SN applications as well as for improving SN performance and security in general. SNADS will provide a cross-platform management of core SN operation. Although SN is built as a synergy of a variety of modules, the system centerpiece is a collection of intelligent agents realizing different computational intelligence and machine learning techniques employed for change detection in signals coming from sensors. The agents may have various levels of intelligence from a simple comparison against the thresholds through the rules system to neural networks structures distributed over the sensor network nodes. The system is written in Java and could be implemented with the JVM technology. The system composition and component design are described. Implementation details are given.

Index Terms—Sensor networks; Signal processing; Java programming

I. INTRODUCTION

This paper describes the software framework integrating computational intelligence and machine learning techniques for signal change detection in signal processing and other related applications. It will explain the development of the Sensor Network Anomaly Detection System (SNADS), which is designated to become a synergetic structure of various intelligent agents, making a change detection decision in signal processing applications. Although the current design focuses on sensor networks (SN) applications, many agents and the framework as a whole are anticipated to be useful in applications far beyond this technology, even for solving problem other than signal processing. An example of using one of its agents for detecting edges in images is given in [1]. The framework is build up as a cross-platform tool with autonomous parts, many of them working independently from each other. The design idea is to make the framework valuable to a great variety of

members of the computational intelligence and machine learning virtual community as well as scholars and professionals who want to employ or wish to investigate a feasibility of applying the intelligent methodologies in their projects. Depending on a particular project's specification, users should be able to choose specific modules and embed them directly in their designs. This will allow for a much faster project prototyping that will cause a significant cost and time reduction. It might also facilitate a feasibility study and a comparative analysis of various intelligent agents in different applications on the implementation level much closer to the final product than the one provided by widely used high level design packages such as Matlab.

Sensor networks have been chosen as an initial designation field because it is an emerging technology with a multitude of important applications in various fields ranging from scientific surveillance to military and security. With the majority of SN employed currently for object and environment monitoring, signal change detection has an ultimate importance in sensor networks. Although a change detection in signals has been investigated for a considerable time, over the last decade there have been new important developments. The literature on change detection is rapidly growing mainly due to applications in engineering, financial mathematics and econometrics. The change detection techniques have developed into various models, which may be classified into likelihood ratio tests, nonparametric approaches, linear model approaches and intelligent techniques [2,3]. Csorgo and Horvath [4] provide a concise overview and rigorous mathematical treatment of methods for a change detection and use a number of datasets to illustrate the effectiveness of the various techniques. Applications of intelligent techniques for personal and technical systems monitoring based on SN applications were reported recently [5-7]. Results received by the first author [8-12] indicate that neural networks and fuzzy models are feasible for description of the kind of uncertainty we may anticipate in SN signals.

SNADS provides support for a wide spectrum of intelligent change detection methods. While triggering alarms based on the limited data provided by the sensors can be accomplished using conventional constructs, the implementations could involve various methodologies and more complicated models, which in turn might require different resources. Depending on the information and resources availability, a variety of models and

Authors are with the Department of Computer Science, Rochester Institute of Technology, 102 Lomb Memorial Drive, Rochester, NY 14623 USA (corresponding author phone: 585-475-7210; fax: 585-475-7100; e-mail: lr@cs.rit.edu).

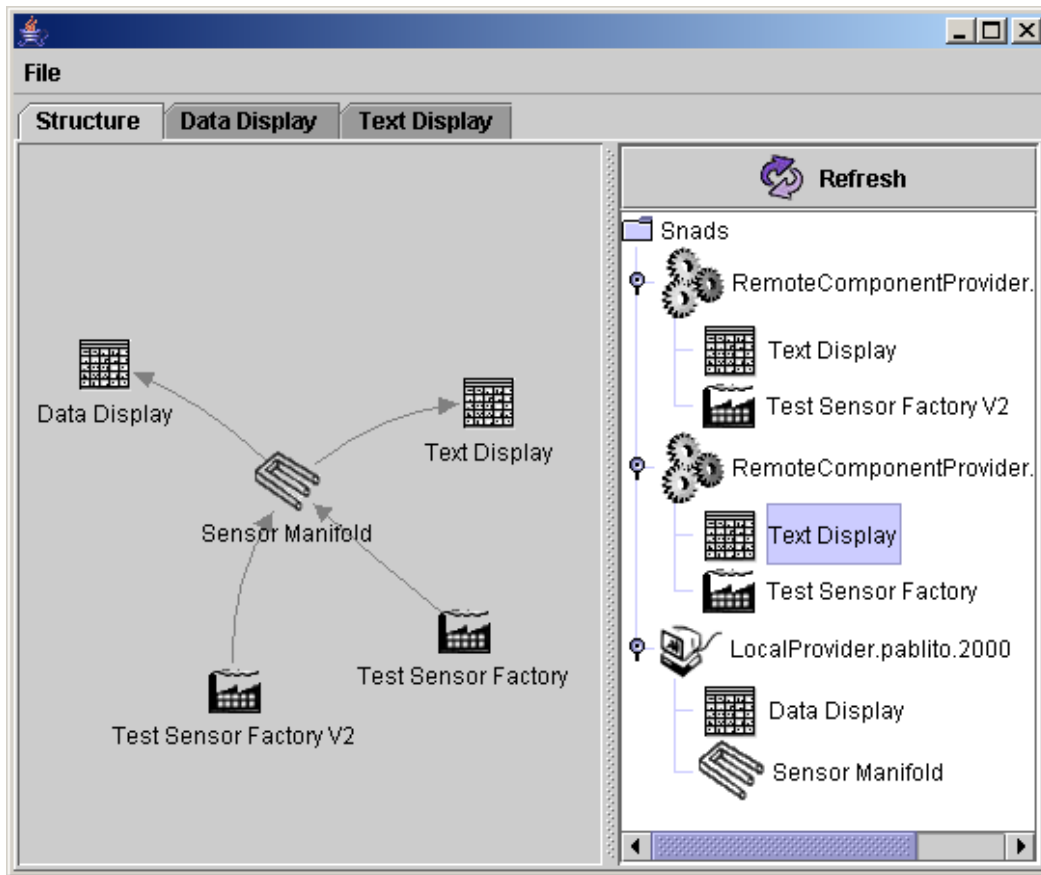


Fig. 1. Graphical system design by component reconfiguration and connection

methodologies could be used in a particular application. In one case, the alarms could be triggered according to the IF/THEN/ELSE rules. For another monitoring application, the system can maintain a history of past values so that it can react according to how quickly measurement results change. Finally, a machine learning technique could be used to derive the model dynamically in order to determine the outcome of the change detection. From the implementation point of view, the application execution will be distributed over the network with simple models used on lower levels and more complicated ones on the upper levels. Depending on the resource availability, however, models and utilities might be redistributed over the network in a given implementation. Simple methods like a comparison against the threshold could be embedded in lower MAC level protocols, while machine learning techniques will be built upon applications.

The paper provides a detail information on SNADS design implementation. SNADS includes the following basic components:

- 1) Message producing and routing system (MPR),
- 2) Sensor configuration and management (SCM),
- 3) Graphical user interface (GUI),
- 4) Change or anomaly detection (ADS),
- 5) Database and storage (DB)

Section II provides the design functional specification with a brief description of each component. The basic

implementation features including main data structures used to implement sensor querying and processing their signals are given in section III. System configuration and compilation procedures are referred to in section IV with a conclusion drawn down in section V.

II. SNADS COMPONENTS AND THEIR FUNCTIONALITY

A. Design overview

The SNADS system is designed as modular, extensible, robust, and scalable. By providing a generic sensor abstraction and sensor-definable configuration mechanisms, SNADS allows for simple, secure management of arbitrary sensor networks. By supporting network nodes with different hardware and software configurations, SNADS will be a versatile cross platform tool. Components will be added and removed from the system during runtime, and components can be upgraded to a newer version on the fly. The architecture is scalable allowing for components to be spread across multiple computing devices. SNADS is designed as an event based system that would allow it to function in a real time mode given enough resources. In this sense some architectural design decisions are similar to those taken in TinyOS design [13].

1) Modularity

Modular design functionality is achieved via implementing a central messaging system, which allows components to work

A	B	C	D	E
Mon Aug 23 21:14:38...	Test Sensor 11	0.6480909717711301		
Mon Aug 23 21:14:38...	Test Sensor 0			0.259309084908384...
Mon Aug 23 21:14:39...	Test Sensor 15			0.2556210166052615
Mon Aug 23 21:14:39...	Test Sensor 4	0.193751790333878...		
Mon Aug 23 21:14:39...	Test Sensor 19			0.9592955358315712
Mon Aug 23 21:14:39...	Test Sensor 7	0.113938420372108...		
Mon Aug 23 21:14:39...	Test Sensor 12		0.3287164742832748	
Mon Aug 23 21:14:39...	Test Sensor 14			0.9469381407933739
Mon Aug 23 21:14:39...	Test Sensor 3			0.126441550954103...
Mon Aug 23 21:14:39...	Test Sensor 8		0.245139145585261...	
Mon Aug 23 21:14:39...	Test Sensor 18		0.140351588944454	
Mon Aug 23 21:14:39...	Test Sensor 13		0.4234410836423813	
Mon Aug 23 21:14:39...	Test Sensor 9		0.338670186738546...	
Mon Aug 23 21:14:39...	Test Sensor 5			0.134519641798284...
Mon Aug 23 21:14:39...	Test Sensor 17	0.8445194473827664		
Mon Aug 23 21:14:39...	Test Sensor 2			0.467548716528338...
Mon Aug 23 21:14:39...	Test Sensor 16	0.7872941699097189		
Mon Aug 23 21:14:39...	Test Sensor 10		0.5845415810887126	
Mon Aug 23 21:14:39...	Test Sensor 6			0.6257687822322973
Mon Aug 23 21:14:39...	Test Sensor 11		0.371610641515756...	
Mon Aug 23 21:14:40...	Test Sensor 11			0.709943383205046
Mon Aug 23 21:14:40...	Test Sensor 0		0.079361192836053...	

Fig. 2 Sensor measurement tabular display

together as a number of black-box entries. This setup allows for simple runtime reconfiguration of the SNADS system and minimizes the damage a malfunctioning component can cause.

2) *Extensibility and upgrading*

The messaging system is also designed to be dynamic and highly extensible. Components can be added, removed, or replaced on the fly by the administrator. Updating to a new version of a component can be done without shutting down SNADS or the network. Self-upgrading and self-modifying option is planned but has not been implemented yet.

3) *Scalability*

A SNADS system implemented over a sensor network platform should be scalable to an arbitrary degree in relation to the number of agents included and modules working together within the limitations given by the hardware platforms in use. From a simple test network to a thousand node perimeter monitoring system, the SNADS architecture can handle it. Because of the message passing abstraction, various SNADS components could easily be located on separate machines, and the workload of a single component could be spread across multiple machines.

B. *Component design basics*

1) *Message Producing and Routing*

The message routing subsystem supports communication between modules in the SNADS architecture and by this way provides the functionality of the whole system in a real time mode. By registering with this system, a component will be able to send events to and receive events from other SNADS components. It will also support workload distribution over networked processors.

2) *Sensor Configuration and Management*

The sensor management subsystem generates and maintains a collection of JSensor objects, which provide the basic abstraction for sensors in the network. The sensor manifold object directly handles all incoming sensor notifications and prepares them for use by other components such as the ADS or GUI. This is the basic unit responsible for getting signals from the monitored objects and the environment and their possible preprocessing.

3) *Graphical User Interface*

The SNADS interface allows for simple central monitoring and management of the system itself as well as of the monitored object and the environment. It works in collaboration with other components and first of all, SCM. Sensors can be dropped from the network or individually configured. Sensor specific configuration dialogs integrate seamlessly into the rest of the GUI. Alerts from the ADS are reported to the GUI, which notifies administrators of potential problems. Additionally, the GUI supports database, session, and ADS configuration.

The GUI allows SnadsComponents to be “wired” together in a way which captures the manner in which data flows through the system intuitively. Components are dragged from their component providers, on the right hand side (see fig. 1), into the working area on the left. There they may be dragged around and connections made by right clicking on a component and choosing, for example “Send Events To” on a Sensor Manifold and then clicking on a Data Display component to create an Event Source / Event Listener relationship between the two. These “wires” may only be connected in ways allowed by the system. A trick that is currently being implemented but not yet finished is creating an

```

C:\Documents and Settings\Jeff\Desktop\sensor project>
9500 cycles remaining - Error = 0.004206764661066625
9000 cycles remaining - Error = 0.004103880390254901
8500 cycles remaining - Error = 0.0040079840398694186
8000 cycles remaining - Error = 0.00391832398046051
7500 cycles remaining - Error = 0.003834256803517336
7000 cycles remaining - Error = 0.0037552281051721603
6500 cycles remaining - Error = 0.003680757278835809
6000 cycles remaining - Error = 0.003610425364992244
5500 cycles remaining - Error = 0.0035438652578098687
5000 cycles remaining - Error = 0.0034807537469671527
4500 cycles remaining - Error = 0.00342080500185271
4000 cycles remaining - Error = 0.0033637651991984324
3500 cycles remaining - Error = 0.0033094080644567284
3000 cycles remaining - Error = 0.003257531148857831
2500 cycles remaining - Error = 0.0032079527029403366
2000 cycles remaining - Error = 0.003160509036871745
1500 cycles remaining - Error = 0.0031150522804991505
1000 cycles remaining - Error = 0.003071448473548761
500 cycles remaining - Error = 0.0030295759299984863
0 cycles remaining - Error = 0.0029893238313158633
Training finished
Serializing network to file...
C:\Documents and Settings\Jeff\Desktop\sensor project>

```

Fig. 3. Neural network training display

“attraction” between the newly created, but not yet connected, end of one of these wires and a component to which it may be connected, and a repulsion away from any component that cannot be connected. The system would in this way provide a subtle cue to the user as to what connections would be wise.

While the GUI elements in figure 1 are standard to the distributed SNADS management, figure 2 shows a simple tabular data display implemented by an external component. In this particular example the Data Display tab was sent from a component on a remote machine, and its content is determined entirely remotely. This was not simple to achieve in Java’s RMI scheme. It was not possible to render entire swing or remote objects because of the restrictions on the creation of dynamic proxy classes underlying the RMI system.

If the GUI component is an instantiated remote object it cannot be marshaled across the network through the RMI system without being replaced by a proxy, but if it is not then the SNADS component behind it cannot communicate to it. The current solution was to send a reference to the GUI component class across the network, creating a new instance (with the class loading handled by RMI automatically) and then calling methods on both the SNADS component and the GUI component to pass the remote references to one another. There may be a way to force a remote object to be marshaled in the normal way.

4) Change or Anomaly Detection

At the heart of SNADS lies the ADS. While a default component will be supplied, arbitrary user-defined ADS implementations can easily be used instead of the default. Based on sample data, the ADS looks for changes in the coming data streams, marks such data, and sends notification to the messaging center. These changes might originate from the actual signal novelty as well as erroneous or maliciously altered incoming sensor readings, ADS is designed as a collection of different agents of various intelligence levels and complexity. Examples of the implemented procedures include a comparison against fixed thresholds, an If/Then rules system, and a multilayer perceptron (MLP) neural network.

The Neural Network implementation used for this project utilizes an open source Java Object-Oriented Neural Engine (JOONE). JOONE appeared to be the best offering among the freely available neural network development tools for the Java language. JOONE also provides a graphical framework for testing neural network architectures [14]. The classes provided by JOONE can be utilized to implement complex and highly scalable neural networks. In the test cases a three layer MLP feed forward neural network has been implemented. The MyNeuralNet class builds this neural network, trains it according to a provided training set, and serializes it to a file for use by SNADS. MyNeuralNet implements a JOONE defined interface which enables an event triggered methodology that is helpful for displaying the status of the training phase.

The MyNeuralNet constructor creates an instance of the JOONE NeuralNet class. MyNeuralNet.Build defines the network’s makeup (e.g. number of inputs, number of outputs, number of layers, number of nodes per layer, type of transfer functions, etc.), reads a training set from an external file, trains the network according to this training set, outputs training progress to the console and serializes the network to another file. The process needs be performed only once as it is this serialized neural network that is needed by the server program, although there is no harm in re-executing the program (and regenerating the serialized network) and in fact this is exactly what should be done if there is a need to change the behavior of the network. Figure 3 presents the neural network training process.

5) Database

The database subsystem provides a simple interface for data logging and searching. Each session is stored separately in the database and the format for sensor readings can be different across sessions. Data cataloged by the SNADS database system is easily accessible for later analysis if so desired. Depending on the specific SNADS application, however, the database may not be used at all.

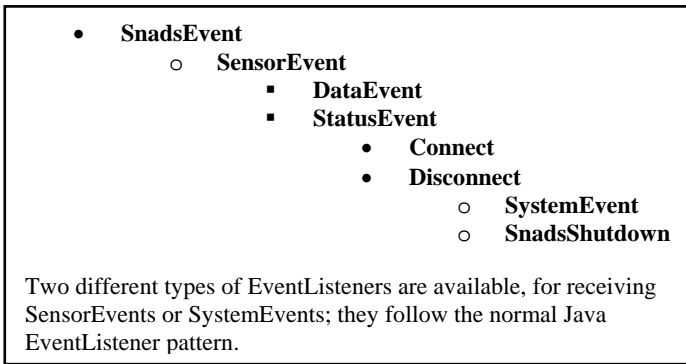


Fig. 4. Sensor event classes hierarchy

Database access is currently provided via the SnadsDatabase class. This will probably become an interface in a future revision so that various underlying database architectures can be more easily supported. At the moment, a MySQL database is used as the backing repository. When the database object is created, a network location for the database is specified. To connect, a username and password must be supplied along with a desired session name. The session name will name a table in the database to hold all readings in the session.

The MySQL database needs a pre-configuration in order to be used by SNADS. Eventually, this will be handled in the GUI database setup dialog. The program expects to use a database called “sensor”. Sensor must contain a (probably empty) table called “templateA” from which it instantiates the session tables. Once the entire GUI is in place, it should be relatively easy to create tables that support different kinds of sensor readings. The templates will be fully editable in the GUI. Furthermore, a table called metadata will be added to the database setup to correlate specific session tables with the template they used.

III. SNADS IMPLEMENTATION FEATURES

A. Current state and future development

1) Java RMI implementation

The SNADS distributed framework is implemented using Java built in RMI methods, in java.rmi package. Interfaces were extracted from the original SNADS classes and modified to fit Java’s RMI [15], which uses dynamic proxy classes in its own implementation, limiting it to the methods of interfaces, rather than both interface methods and class methods.

SNADS class hierarchy (see fig. 4 for events classes hierarchy) has been designed to better fit the distributed system and to allow for more generalized object management. Every class that would be accessed over the network was made a subclass of SNADS component, allowing for generalized object management and user interaction, both through a standardized configuration method and through standardized methods for the retrieval of GUI objects. The basic SNADS component class also implements (via the Named interface) methods for the identification and retrieval of components.

The ComponentProvider forms the basis for both the administrative console and the remote units. They are registered with Java’s RMIRegistry, allowing the client to find them automatically. The client retrieves a set of components made available by a given provider to present to the user. Some components are only available once, such as a specific SensorFactory on a specific remote system, while others, such as BasicSensorManifold, may be instantiated multiple times. The current MultiComponent implementation leaves something to be desired, as it must create an instance of the object in order to offer it to the user, despite the fact that it may never be selected by the user for inclusion in a sensor system. If the creation of the object were to involve threads on a remote server, database connections or a large memory utilization would waste resources.

2) Partially Implemented Features

The message processing system is still in an infantile state. All message routings are currently hard coded, which defeats many of the system goals. Unfortunately, this system was not implemented first, as it should have been, but instead it was added after a minimally functional system had been created. The next milestone for SNADS involves properly implementing the message system to support dynamic message routing and type addition. A couple of database features are also missing at present. Specifically, the ability to select a sensor reading format template is not present, but the underlying code is in place. The change detection by the ADS is not yet indicated in the GUI. However, it is properly stored in the database. Almost everything is in place for the GUI to handle these events, but a last minute rethink of the setup prevented inclusion in this release.

3) Known Bugs

For some yet to be determined reason, the SNADS system (or at least the GUI) appears to hang when around 30 sensors are added to the network. Obviously, this is unacceptable and will be fixed in the next milestone.

4) The Next Step

The next major milestone for SNADS will include full GUI support for configuration of all major subsystems, a couple of bug fixes, and finalization of the messaging system. Also slated is the dynamic creation/destruction of sensor factories and a number of extensions in the database interface. Once anomaly reporting is fully integrated into the GUI, the first public release will be offered as v1.0.

B. The JSensor Abstraction

In order to support diverse sensor platforms, SNADS has a generic sensor abstraction. JSensor is an abstract class definition which provides functionality to disseminate incoming data and sensor status events. The basic JSensor also provides support for unique per session identification. Classes derived from JSensor must also support queries for sensor name, type, and capabilities as well as provide a configuration mechanism. Optionally, such classes may support sensor shutdown and renaming operations. By default, attempts to issue these commands result in an exception.

1) Session IDs

A session ID is assigned to each JSensor object via its constructor. Each ID issued must be unique within the current

session in order to identify each sensor properly. Currently, sensor factories are charged with managing ID generation, however this may change in the future.

2) *Sensor Name and Type*

Each sensor has a name and type associated with it. There are no restrictions placed on these fields and they can be used by extending classes as they see fit. However, some standards may help. The type field should contain information about the actual kind of sensor, which the JSensor represents. For example, a type query for one of the simulated sensors used in testing would return a string like "TestSensor;vers=1.2;". Names do not have to be unique, but should describe the purpose of the sensor.

3) *Sensor Capabilities*

Sensor capabilities specify the non-standard functionality, which a sensor supports. A capability is specified in *Interface.method()* format. The standard capabilities are JSensor.shutdown() and JSensor.setName(); these do not have to be supported. As an example of extending the basic functionality set, the test sensors added TestSensor.skew() and TestSensor.unskew(). While these methods were only called from within the TestSensor configuration dialog, a SNADS module could theoretically utilize these methods directly after querying for them dynamically.

4) *Sensor Configuration*

The JSensor class has an abstract method called configure, which must be supported by deriving classes. All the method has to do is allowing for configuration of a specific sensor. Theoretically, if the sensor has no configuration options, this could result in no operation. While not a requirement, standard configuration methods create a GUI interface for a dynamic configuration. Any other approach should be well documented. The configuration procedure may be more strictly defined in a future revision of the SNADS specification.

C. *Sensor Factories*

Because of the varied underlying technologies, there is no single way to instantiate a new JSensor derived class. As such, each type derived from JSensor should have a corresponding sensor factory class, which implements the JSensorFactory interface. Basically, the sensor factory just waits for new sensor connections, creates a JSensor derived object, and adds it to the sensor manifold.

D. *Error Handling*

In the operation of the distributed network it is possible for entire nodes to become disconnected, leaving the proxies connected to Remote objects, talking to so much dead hardware. Originally the system was set to detect these events proactively and at the earliest possible time and propagate the appropriate LostComponent events through the messaging system. This required each component to listen for these events, and check the component identified as lost against all member variables of type SnadsComponent and members of any Collections, Arrays and the like used by the object. This system quickly grew to the size of the rest of the program, and added much complexity, as for example, in handling one lost

component the system could come across another lost component, triggering another LostComponent message.

The replacement scheme used instead is not centralized, and does not rely on the messaging system. Every method of a Remote interface must be declared to throw a RemoteException, which will occur when the method is called on a component located on an unreachable node. At that time only the component triggering the exception is expected to remove its own references to the lost component. In this way each object deals with its loss on its own schedule, as it needs to. The code for dealing with the loss of an object is located where the object is used, so the object catching the RemoteException does not have to search through a number of locations to find the references. If the author of a future component wished to locate all lost objects' code in a single method, he could call that method whenever a RemoteException is caught.

IV. SYSTEM CONFIGURATION AND COMPILATION

A. *Network Configuration Persistence*

The network structure created in the work area of the management GUI can be saved to disk and reloaded later. Using the named interface of the components and the providers, they came from, a fully qualified component name is constructed and saved. During loading the system looks for the same component provider (by name) and gets the same component (again by name) from it. In the event that the specific provider is not available the system will search all providers until it finds a matching component.

Currently version information is a part of a component's name. It would be better to store this separately so that the loading process could load a newer version of a component if it has been upgraded. If a component is not available at all, the load should either fail or load the network with placeholders for the missing pieces. This state is not currently handled.

B. *Component Upgrade*

To integrate a new version of a component into the system either a RemoteComponentProvider must be restarted or a new one started. Hitting "refresh" on the console will show the new information.

C. *Compilation*

The SNADS distributed framework was developed and compiled under Eclipse Version 2.1.3 built with the RMI Plug-in for Eclipse version 1.5.1.1 from <http://www.genady.net/rmi/>. This RMI plug-in is a commercial software, however its only purpose was to add the RMI Stub generation for certain classes to the Eclipse build process without any effort. It is possible to build the SNADS distributed framework using only Java's command line tools.(See for details on Rmic

<http://java.sun.com/j2se/1.3/docs/tooldocs/win32/rmic.html>)

Java RMI has the following known drawbacks.

- Need for base class and interface, i.e. SNADSComponent and BasicSnadsComponent
- Need for RMI compiler to create subclasses, which must be sent over the network and loaded via a class

loader at run time, even though RMI uses only interfaces.

- Difficult to send a real object to another machine and then obtain a remote interface for it, as in a GUI.

V. CONCLUSION

SNADS is designed as a synergetic collection of heterogeneous computational intelligence and machine learning agents. Its dynamic structure allows for adding up new modules realizing new intelligent techniques previously unavailable and scaling it up to really big numbers of the modules and agents. It is realized in Java that allows for its application with novel Sun Worldspot sensor kits currently available on the market [16].

While still in its early stages, SNADS demonstrates a great deal of potential for the future developments and applications. By facilitating general sensor network security and management even across heterogeneous networks, such tasks will become more accessible. There is an obvious set of scenarios, in which SNADS would be of great use, especially for objects, processes and environment monitoring. By allowing a user to more quickly and easily create SN configurations, SNADS will be invaluable to scientists using sensor networks to monitor objects and the environment. Additionally, the ADS elements can help to identify results that need to be more closely scrutinized to ensure that the data set is not polluted by errors or attack.

Besides the obvious applications in traditional sensor networks, SNADS has a high potential in other areas as well. For example, traditional computer intrusion detection systems could be augmented/replaced by a network of simulated sensors. Each sensor would sit on a separate processor and report statistics back to the SNADS processing center.

REFERENCES

- [1] Reznik L., Von Pless G., and Al Karim T. "Application testing of novel neural network structures" Workshop on Building Computational Intelligence and Machine Learning Virtual Organizations, Fairfax, VA, October 2008
- [2] Xiaolong D. and Khorram S. "Development of a new automated land cover change detection system from remotely sensed imagery based on artificial neural networks", IEEE International Geoscience and Remote Sensing, 'Remote Sensing - A Scientific Vision for Sustainable Development', 1997, 3-8 Aug. 1997 Singapore, vol.2, pp.1029 - 1031
- [3] Han M., Xi J., Xu S., and Yin F.-L. "Prediction of chaotic time series based on the recurrent predictor neural network", IEEE Transactions on Signal Processing, vol. 52, Issue 12, Dec. 2004 pp.:3409 - 3416
- [4] Csorgo M. and Horvath L. "Limit Theorems in Change-Point Analysis" New York: John Wiley & Sons, 1997
- [5] Majeed B., Nauck D., Lee B.-S., and Martin T. "Intelligent systems for wellbeing monitoring", Proceedings of 2nd International IEEE Conference on Intelligent Systems BT Exact Intelligent Syst. Lab. Res. & Venturing, British Telecom plc, Ipswich, UK, 22-24 June 2004, Vol.1, pp. 164 - 168
- [6] Shan Q., Liu Y., Prosser G. and Brown D. "Wireless intelligent sensor networks for refrigerated vehicle", Proceedings of the IEEE 6th Circuits and Systems Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication, 2004, 31 May-2 June 2004, Vol.2, pp. 525 - 528
- [7] Reznik A.M., Shirshov, Yu.M. Snopok, B.A. Nowicki, D.W. Dekhtyarenko A.K. "Associative memories for chemical sensing" Proceedings of the 9th International Conference on Neural Information Processing, ICONIP '02, vol.5, 18-22 Nov. 2002, pp. 2630 - 2634
- [8] Reznik L. and Dabke K.P. "Evaluation of Uncertainty in Measurement: A Proposal for Application of Intelligent Methods" in H. Imai/Ed. Measurement to Improve Quality of Life in the 21st Century, IMEKO - XV World Congress, June 13-18, 1999, Osaka, Japan, vol. II, p.93-99
- [9] L. Reznik and Pham B. "Fuzzy Models in Evaluation of Information Uncertainty in Engineering and Technology Applications", The 10th IEEE International Conference on Fuzzy Systems, December 2-5, 2001, Melbourne, Australia, vol.3, pp.972-975
- [10] Reznik L. and Dabke K.P. "Measurement Models: Application of Intelligent Methods", Measurement, vol. 35, No.1, pp.47 - 58, 2004
- [11] Mari L and Reznik L. "Uncertainty in Measurement: Some Thoughts about its Expressing and Processing", In L.Reznik and V.Kreinovich/Eds. *Soft Computing in Measurement and Information Acquisition*, Springer, Berlin-Heidelberg-New York, 2003, ISBN 3-540-00246-4, pp. 1-9
- [12] Reznik L. Which models should be applied to measure computer security and information assurance? Proceedings of the FUZZ '03, The 12th IEEE International Conference on Fuzzy Systems, St.Louis, May 25- 28, 2003, IEEE, vol. 2, pp. 1243-1248
- [13] TinyOS Community forum at <http://www.tinyos.net/>, retrieved on September 1, 2008
- [14] JOONE - Java object oriented neural engine, <http://www.jooneworld.com/>, retrieved on September 1, 2008
- [15] D. Kurtis "Java, RMI, and Corba", White paper, available at <http://www.omg.org/library/wpjava.html> retrieved on September 1, 2008
- [16] Project Sun Spot at <http://www.sunspotworld.com/products/>, retrieved on September 1, 2008.