A Unique Solution for Designing Low-Cost, Heterogeneous Sensor Networks Using a Middleware Integration Platform

Jarrod Trevathan, Trina Myers

***Abstract*—**Proprietary sensor network systems are typically expensive, rigid and difficult to incorporate technologies from other vendors. When using competing and incompatible technologies, a non-proprietary system is complex to create because it requires significant technical expertise and effort, which can be more expensive than a proprietary product. This paper presents the *Sensor Abstraction Layer* (SAL) that provides middleware architectures with a consistent and uniform view of heterogeneous sensor networks, regardless of the technologies involved. SAL abstracts and hides the hardware disparities and specificities related to accessing, controlling, probing and piloting heterogeneous sensors. SAL is a single software library containing a stable hardware-independent interface with consistent access and control functions to remotely manage the network. The end-user has near-real-time access to the collected data via the network, which results in a cost-effective, flexible and simplified system suitable for novice users. SAL has been used for successfully implementing several low-cost sensor network systems.

***Keywords*—**Sensor networks, hardware abstraction, middleware integration platform, sensor web enablement.

I. Introduction

A plethora of sensing technologies has appeared with the emergence of sensor networks. Because of hardware incompatibilities and a lack of standards, difficulties have arisen in the ability to integrate multiple sensing devices from different manufacturers within the same network. While each technology has its own benefits, there is a strong motivation for a user to be able to mix these technologies within the same architecture to make the most of their respective strengths. This parallels the problems faced by early operating systems where users demanded the ability to attach peripheral devices (i.e., mouse, keyboard, printer, joystick, etc.) from competing vendors without the need to have to engage in a complicated configuration process. One of the best approaches towards achieving the goal of a plug'n'play sensor network is to use a middleware solution.

Middleware refers to software that sits between the hardware and the higher-level application software and it facilitates the communication between these different technologies and systems. Sensor and instrument middleware is an active research area [1]-[9]. The main requirement is that middleware must be generic and not tied down to specific sensor/instrument technologies (to reduce costs). Commonly, hardware abstraction is fully integrated with the middleware software [10].

This paper proposes the *Sensor Abstraction Layer* (SAL) – a unique platform for developing low-cost heterogeneous sensor networks. SAL allows a sensor network to use technologies from multiple vendors to create a purpose-built and typically less expensive system. SAL is a *middleware integration platform* [11], which manages and abstracts communications and interactions with sensor hardware. In a middleware stack, SAL sits at the bottom, close to the sensors it manages, and proxies all communications with these sensors (Fig. 1). The upper middleware layers rely on the generic interface provided by SAL to access and control sensors. This interface can be used without the knowledge of technology specifics, provided that sensor technologies device plug-ins are registered with SAL (analogous to attaching a peripheral device to a personal computer).

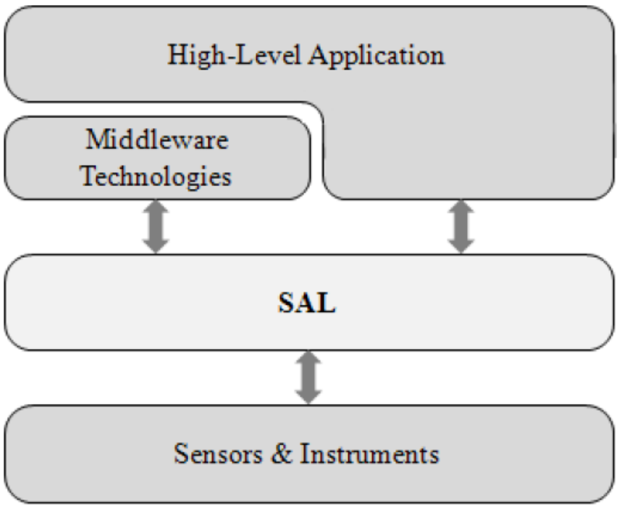

Fig. 1 The SAL Software Model

There are no existing solutions that support the creation of low-cost heterogeneous sensor networks in current literature. Therefore, the concept underpinning SAL is novel. SAL has been successfully trialled in a sensor network system that was deployed on the Great Barrier Reef [12]. SAL is also being

J. Trevathan is with the School of Information and Communication Technology, Griffith University, Brisbane, Queensland, 4111, Australia (phone: 6107 3735 5046; e-mail: j.trevathan@griffith.edu.au).

T.S. Myers is with the School of Business (Information Technology), James Cook University, Townsville, Queensland, 4811, Australia (phone: 6107 4781 6908; e-mail: trina.myers@jcu.edu.au).

employed by the *Smart Environmental Monitoring and Analysis Technologies* (SEMAT) sensor network system [13], [14]. Finally, SAL is being used for a "smart home" initiative to monitor household energy consumption. In each case, SAL was able to create a sensor network system that was dramatically less expensive than existing generic proprietary solutions.

This paper is organised as follows: Section II provides background on existing sensor network middleware solutions and emerging standards. Section III gives an overview of SAL's functionality and internal architecture. Section IV describes how SAL is used to manage a sensor network with heterogeneous technologies. Section V discusses the XML standards and sensor technologies supported by SAL. Section VII describes sample applications where SAL has been successfully used. Section VIII provides some concluding remarks and avenues for future work.

II. Related Work on Middleware and Sensor Web Enablement standards

A. Sensor Network Middleware

There have been various proposals for constructing sensor network middleware. The following describes some of these proposals.

Cougar Bonnet et al. [15] adopts a database approach where sensor readings are considered to be in "virtual" relational database tables. An SQL-like query language is used to issue tasks to the WSN. First concrete experiments show that even very simple protocols and algorithms can exhibit surprising complexity at large scale.

Mate´ is an architecture for constructing application-specific virtual machines that executes on top of TinyOS [5]. Developers can easily change instruction sets, execution events, and virtual machine subsystems using this architecture. Mate´ provides a simple programming interface to sensor nodes. For example, a sense-and-send program can be written with six instructions.

Impala is a middleware designed for use in the ZebraNet project, supports control in the application itself by exploiting mobile code techniques to change the functionality of the middleware executing at a remote sensor [6]. The key to energy efficiency for Impala is for the sensor node applications to be as modular as possible, enabling small updates that require little power during transmission.

MiLAN (Middleware Linking Applications and Networks) has an architecture that reaches the network protocol [3]. MiLAN is intended to sit on top of multiple physical networks. It acts as a layer that allows network-specific plug-ins to convert MiLAN commands to protocol-specific ones that are passed through the usual network protocol stack. Therefore, MiLAN can continuously adapt to the specific features of whichever network is being used in the communication.

Mires is a publish/subscribe middleware with the goal of reducing the overhead of passing messages up through the network [8]. Mires takes a message-oriented approach due to the low availability of resources and processing capacity of sensor nodes. Applications can subscribe to data sources (i.e., sensors). When information becomes available, a data source publishes it and all subscribers are then free to view the data.

Hasiotis et al. [11] take a high-level approach, where a sensor network is regarded as a source of information similar to a relational database. The generic API offers abstracted methods to query devices and to locate them. However, there is no support for automated detection and sensor-specific features. Handziski et al. [16] present the *Hardware Abstraction Architecture* (HAA), which uses a three-layered software stack that is implemented on MSP430 embedded platforms running TinyOS[1]. The bottom layer deals with raw hardware and offers a basic set of methods. The methods become more and more generic towards the top layer. To deal with the many hardware disparities, HAA maintains a complex set of interfaces and a versioning system, and resorts to software emulation to compensate hardware deficiencies.

The *Linked Stream Middleware* (LSM) initiative [17] is a platform that brings together the live real world sensed data and the Semantic Web in a unified model. The LSM provides an extensive range of functionalities: different wrappers to access stream sources and transform the raw data into linked-stream data; data annotation and visualisation; and live querying over unified linked data.

The challenges in designing and implementing WSN-middleware include the conflict between distributed computing and embedded sensor devices, the degree of application-specific requirements, and the *Quality of Service* (QoS) [18]. Distributed computing should support scalability, reliability and heterogeneity when designing dynamic network topologies. To support and optimize a broad range of applications, may lead to compromises in functionality versus efficiency. The QoS of various applications must be considered because the limited resources affect the performance requirements of all running applications [18]. Here we discuss the development of SAL and the alignment to these challenges.

B. Sensor Web Enablement (SWE)

The *Open Geospatial Consortium* (OGC) *Sensor Web Enablement* (SWE) group [19] are developing common standards to:

- Discover sensor systems observations and observation processes;
- Determine a sensor's capabilities and quality of measurements;
- Access sensor parameters, allowing software to process and geo-locate observations;
- Retrieve real-time or time-series observations and coverage in standard encodings;
- Task sensors to acquire observations of interest; and
- Subscribe and publish alerts issued by sensors or sensor

[1] http://www.tinyos.net/

devices based on certain criteria.

This common interface is achieved by developing a standard XML-based web service that can be invoked in either a REST or SOAP/WSDL methodology. The goal of providing a web service interface is to create a SensorWeb, and to begin standardizing the discovery and communication protocol between various heterogeneous sensors. The process begins with sensors seamlessly integrating other sensors together into a global network.

Where possible, SAL's architecture adheres to the SWE standards to harness the aforementioned desirable qualities of a sensor web enabled system.

III. Overview of SAL

A. Design Goals

The design goals for SAL are as follows:

- Create a simplified middleware architecture, which allows the sensors to be more "portable" by removing technology dependent code [2];
- Create a low-overhead split design, which allows SAL to run on limited processing-power embedded platforms;
- Provide guaranteed compatibility and interoperability with future middleware technologies by using emerging standards as defined by SWE;
- Provision for basic data processing, sensor auto-discovery and configuration, and local caching of data to account for unreliable network links;
- The ability to turn any source of information into a virtual sensor (or "pseudo sensor"), which is fully integrated into the rest of the sensor network;
- Provide a two-way communication channel with sensors so sensor-specific commands can be issued and their results collected and passed up the hierarchy;
- Provide remote real-time control over sensors and near-real time streaming of data collected from sensor devices; and
- Create an inexpensive alternative to proprietary systems with the ability to mix and match hardware from different vendors, and the capacity to run on commonly available computing devices with limited computational capabilities.

For the purposes of this paper, SAL is written in Java and is implemented on a Linux-based operating system.

B. SAL Functionality

The functionality provided by SAL can be grouped in three distinct categories:

1. Hardware management;
2. Hardware discovery; and
3. Hardware control.

Hardware management functions allow users to manage sensor networks under SAL's supervision. Typical management tasks include adding and removing sensors, editing and changing a sensor's configuration, and setting up special channels for real-time reports on hardware status.

Hardware discovery functions allow users to express search queries to find sensors matching a specific criterion. To this end, SAL maintains a SensorML document for each sensor (see Section V *A*). These documents are used to resolve lookup queries (among other things). Hardware discovery functions are also used to find out about hardware capabilities (such as sensor nodes and gateways).

Hardware control functions allow users to *pilot* sensors (i.e., control/operate sensors). SAL enhances sensor capabilities by providing transparent support for any sensor-specific commands. To achieve this, SAL encapsulates sensor-specific commands into generic ones. Generic commands have a unique format, which is sensor-technology independent. This way, users wanting to pilot a sensor need not worry about the sensor technology of the node. They only need to invoke a generic command and SAL will then translate it into the appropriate sensor-specific one.

C. The Client/Agent Architecture

SAL's software is divided in two separate parts to produce an efficient and scalable application capable of running on resource-limited sensor gateway platforms:

1. A SAL agent; and
2. A SAL client.

An agent connects directly to the sensor hardware. The user interacts with the client (via a user interface) to control and receive data from the agent. There can be potentially many agents within the system each controlling its own set of sensor devices. There is a one-to-many relationship between a client and agents. A single client can manage and support multiple agents, but an agent can only be managed by one client (Fig. 2 (A)).

1) SAL Agents

The SAL agent runs on the sensor gateway platform. This is the core feature of SAL, which implements only essential low-level sensor-access related functions. The agent handles the details involved in communicating with the hardware and must therefore have the appropriate software stack required to establish this communication. In a typical scenario, an agent registers a client on start up. It then waits for sensor commands from the client, executes them and returns a result. The agent setup is suitable for operation on low-powered and low-cost computer devices.

Figs. 2 (B) and (C) illustrate the extended scenarios that the agent/client setup up offers. Each agent can have its own array of heterogeneous sensor technology for which it is responsible. In turn, the client can control multiple agents, each of which can be different types of computing devices with different hardware specifications (Fig. 2 (B)). Alternately, agents can in turn be used as intermediate sources of information where one agent treats another agent as if it was a sensing device in its own right (Fig. 2 (C)). This is useful in the situation where an intermediary device with higher computational power or expensive communications capabilities is required to temporarily store and retransmit data on behalf of less capable devices.

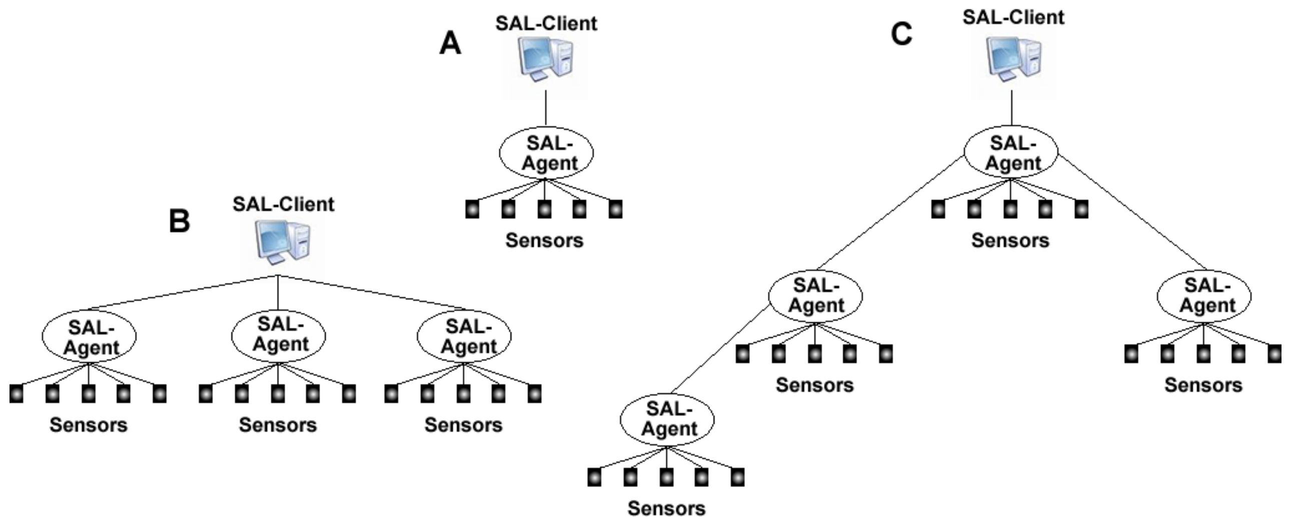

Fig. 2 Scalable SAL Hierarchy involves: (A) one-to-many relationships between one SAL-client and many SAL-agents; (B) clients can control multiple agents, each with different types hardware specifications; and (C) agents can be intermediate sources of information

2) The SAL Client

The SAL client is the user-visible part of SAL. A user can interact with the client via a Graphical User Interface (GUI) or a text-based interface. Essentially, the SAL client is a library of functions, which can be invoked to trigger specific actions on sensors managed by an agent. The following high-level features have been implemented in the SAL client in order to keep the agent lightweight (in terms of storage and computation requirements):

- The client is responsible for scheduling access to sensors.
- Multiple concurrent accesses to the same sensor must be coordinated. To reduce the load and request processing time on the agent merging multiple requests into a single one is sometimes required. The client then duplicates the generated response as many times as required.
- A SAL client also provides sensor search-related functions. Search queries can be sent to an agent to find sensors matching specific criteria. The SAL client can offer basic quality checking and data processing functions. For example, checks can be performed on the raw readings given by a sensor to ensure the reading is valid.
- A client can also process raw readings to extract results that are more meaningful. Information about data processing and quality checking is sensor-specific and is stored in that sensor's SALSensorML document (explained further later).

3) Local/Remote Agents and Communications

There are two scenarios for the placement of agents within the SAL system: local and remote. Firstly, an agent may be local in that the agent and client are both contained on the same device. A local SAL agent instance on a sensor gateway runs in its own Java Virtual Machine (JVM). The SAL agent interface (SAL API) is only available in that JVM. To use the interface, a client application must be run in the same JVM as the agent. This approach has the added advantage of low-overhead method calls and low latency. Alternately, an agent may be remote in that the agent and client are running on different independent computing devices. A remote SAL agent is an instance of a SAL agent whose interface has been exported using Java Remote Method Invocation (RMI). All methods in the SAL API of a remote agent can be called from a separate JVM over a network connection. In reality, many sensor networks may be a hybrid potentially containing at least one local agent and multiple remote agents.

The client and agents communicate over the network using User Datagram Protocol (UDP). UDP is a connectionless protocol and its datagrams are more compact than the alternative connection-oriented Transport Control Protocol (TCP). UDP is suited to SAL as there is no requirement for the reliable transport of data that is offered by TCP, which introduces high overhead and use of bandwidth. As agents and clients communicate over possibly long distance unreliable network links, agents constantly monitor the state of their connections with the client. The agent stores all data locally if the client becomes unreachable. When network connectivity is re-established, the cached data is uploaded. The cache is then flushed of any outstanding or incomplete data.

4) Use Case

A typical SAL use case is as follows:

1. *A* SAL client creates an instance of a local SAL agent or obtains a reference to a remote agent via the RMI proxy.
2. Whether local or remote, the SAL agent interface is implemented by both references returned by the RMI proxy.
3. The client may want to subscribe to specific events (such addition/removal of sensors).
4. The client obtains a list and description of sensors

managed by the agent to save a couple of method invocations just to stay up-to-date.

5. Loop over the following steps:
   a. The client identifies which sensor to control and asks the agent for a list of capabilities supported by that sensor. This list is merely a collection of commands (and their descriptions) that can be sent to the sensor to trigger a pre-determined behaviour.
   b. The client picks a command and instructs the agent to execute it.
6. The result of the command is sent back to the client.
7. Just before exiting, the client releases the SAL agent and then terminates.

IV. Piloting sensors

This section explains how different sensor technologies are abstracted by SAL so that sensor network software can be technology-independent. How the generic methods in the SAL API can be translated into sensor-specific ones are shown. The low-level details of the hardware communication and control are explained, and then the abstraction process. Then, a detailed description of a SAL agent's software layers is given. Lastly, SAL's automated sensor detection features are discussed.

A. Accessing the Hardware

From a sensor gateway's point of view, enabling communication with sensor nodes can be broken down into a series of steps. This modular approach gives SAL the flexibility to support newer technologies.

- A sensor node typically connects to a computer through a special controller called a native controller. The native controller is responsible for "translating" the sensor's electrical interface into that of the computer's I/O port (an I/O port can use a wireless communication medium).
- Sensor devices based on the same underlying technology share the same electrical interface. The interface defines the required communication medium, how many pins the sensors contain, how the pins are used. The electrical interface also specifies what the resulting topology looks like if multiple sensors can share the same communication medium.
- Access to an I/O port and its connecting physical medium is managed by the port controller. The controller translates the port's electrical interface to that of the data bus, connecting the computer's main components (e.g., RAM, CPU, etc.). Typical I/O ports can be used to attach sensor trees. This includes serial, parallel, FireWire, USB, Ethernet, PCI, i2c and GPIO ports, which are hardware endpoints that allow bits of information to be sent and received over a physical medium.

Notably, this model applies to both wired and wireless sensor nodes. Wireless sensor networks also have their own data sinks, which, typically, can either be network-reachable and queried using a high-level protocol (such as SNMP, HTTP or FTP), or connectable directly to a sensor gateway through an I/O port. This model communicates with a sensor node because the software stack (SAL with the operating system) needs to know how to communicate with all controllers involved. This function is handled by separate software layers in one of the following three ways:

1. The operating system kernel usually includes drivers for common I/O port controllers found on recent platforms;
2. Software code to pilot native controllers can either be included in the kernel (in the form of a driver); or
3. They can be implemented as a user-space application or library of functions.

Regardless of which form of controller, a native controller driver relies on interfaces and methods provided by I/O port controller drivers in the kernel.

The task of sending a generic command (as provided by the SAL API) to a sensor and reading its response is divided into multiple logical blocks. The following four steps are required:

1. The generic command must be translated into a native command the sensor understands.
2. The native command is passed on to the native controller driver, which encapsulates it using the appropriate protocol.
3. The resulting data unit is passed on to the port controller driver.
4. After further encapsulation, the I/O port controller physically transmits the data to the native controller over the physical medium as a series of bits.

The process happens in the opposite order when a sensor generates a response.

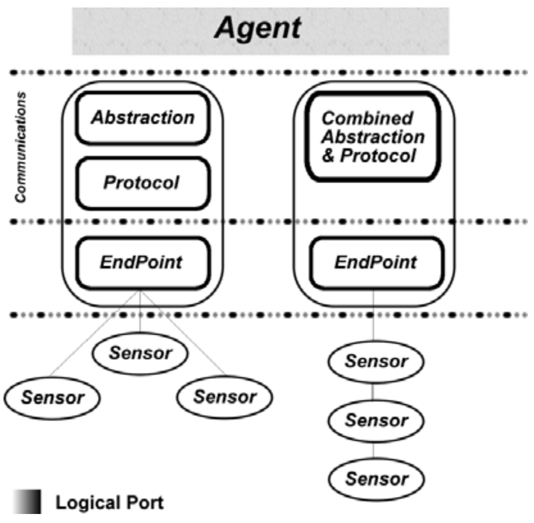

Fig. 3 Software Layers in SAL

B. Sensor Abstraction

Abstracting and hiding hardware-specific details is a complex task. Fig. 3 illustrates how the work is divided among three software layers. The top Agent layer is a simple layer, mainly responsible for handling the communication with SAL clients. It receives packets, parses them, calls appropriate methods in the underlying communications layer, and sends back a response containing the results of the call.

International Science Index Vol:8, No:4, 2014 waset.org/Publication/9998078

The Agent layer also maintains the state of connections with the SAL client, which allows ongoing data streams to be stored in the local cache (also implemented at this level) when network disruptions occur.

The job of the Communications layer is twofold, it provides hardware management methods to configure and set up hardware parts and it provides hardware control methods. These methods translate a generic command (as provided by generic methods in the SAL API) into one that can be transmitted to and understood by a sensor's native controller (referred to as a native command).

To achieve this, the Communications layer is divided into two sub-layers, the Abstraction sub-layer and the Protocol sub-layer. The Abstraction sub-layer acts as an adapter layer. It implements generic methods matching the set of generic commands provided by SAL. Each generic method then calls appropriate sensor-specific methods provided by the underlying Protocol sub-layer to carry out the tasks set in the generic command. In contrast, the Protocol sub-layer is sensor-dependent because it implements the native communication protocol used to "talk to and address" native controllers and their attached sensors. Software code at this layer is usually provided by a third party (e.g., the device vendor, an open source project, etc.), although it could be implemented from a specification document in a homemade software block when the communication protocol is trivial (serial devices for instance). When a generic command is received, the SAL Agent layer parses it and calls the appropriate abstraction layer method matching the generic command. The abstraction sub-layer acts as an adapter to the protocol layer and translates SAL generic methods into sensor-specific ones.

At the bottom of the stack is the EndPoint layer. This layer is tightly coupled to the I/O hardware ports available on the gateway. This layer takes care of transporting the sensor native commands (as produced by the Protocol sub-layer) to the native controller. Software code at this layer is normally included in the operating system kernel, and SAL only needs to make sure it is available and properly configured.

Fig. 3 illustrates the concept of Logical port. A Logical port combines an endpoint, a protocol, and its abstraction. This port groups the data structure and methods used to communicate with a sensor tree under a single element. Each sensor is associated with a Logical port. SAL uses the Logical port's generic methods to communicate with the sensor.

1) Advantages of the Abstraction Approach

Splitting the task of communicating with sensors presents three main advantages: decreased configuration effort, layering to promote independence of tasks, and remote accessibility. These stem from the fact that the aforementioned layers naturally follow the hardware and software boundaries existing in a Linux operating system.

Firstly, the amount of effort required to add support for a new instrument technology is minimal and is only a matter of creating a Logical port; the EndPoint layer software is found in the kernel. Software at the Protocol layer, which is specific to sensor technology, is widely available, usually in the form of device drivers, user-space libraries or programs. At the Abstraction layer, a simple adapter must be written to wrap the protocol layer software and provide generic methods.

The amount of code required to write for a new technology only depends on how many generic methods are exposed by SAL. In very few cases, there is no pre-existing protocol-level software because a trivial one can be easily created from the device's specifications. Here, a homemade implementation spans both the Protocol and the Abstraction layers. This implementation provides a SAL generic interface (Abstraction layer) and translates them into direct calls to the low-level function in the EndPoint layer. Communications over serial ports are an example where an implementation can be written to fill in the gap.

Secondly, configuring the logical ports is made easier because each layer carries out specific tasks independently of each other. Each EndPoint and Protocol block in a logical port has its own parameters that control communications. These parameters are described in a document referred to as a platform configuration document (see Section V *C*).

Finally, this model allows literally any source of information accessible to the platform (remotely controllable or not) to be turned into an instrument manageable by SAL. These are referred to as pseudo sensors. For example, SAL's communication layer includes support for the Simple Network Management Protocol (SNMP) so that objects managed by an SNMP agent appear as sensors. In the same fashion, operating system state information such as load average, idle/system/user time and free memory is also made available in the form of pseudo-sensor fully integrated with the rest of the network.

C. Automated Sensor Detection

Runtime automatic sensor detection and configuration (also known as hot-plug) is highly hardware dependent. One direct consequence of the communication model explained in the previous subsection is that sensor auto-discovery is possible only when both port and native controllers allow it.

Some I/O ports, such as USB and IEEE 1394/Firewire, support hot-plug natively without the need to reboot the platform. Newly connected native controllers are detected and reported by the operating system. In most cases, if a driver is available, it will also be loaded in the operating system. Some other I/O ports support hot-plug but do not advertise connected devices (serial ports for instance), while other I/O ports may require the platform to be restarted altogether (e.g., PC104 ports).

Depending on the native controller (and therefore on the sensor technology too), SAL may not yet be able to use the newly connected sensors. Some sensor (1-wire sensors for instance) advertise themselves so they can effectively be discovered by SAL and used straight away. However, others (e.g., analogue sensors connected to a LabJack I/O USB converter) cannot be discovered since the native controller (i.e., the LabJack interface) does not allow sensor polling and discovery. In this case, manual intervention is required in order to use the sensors.

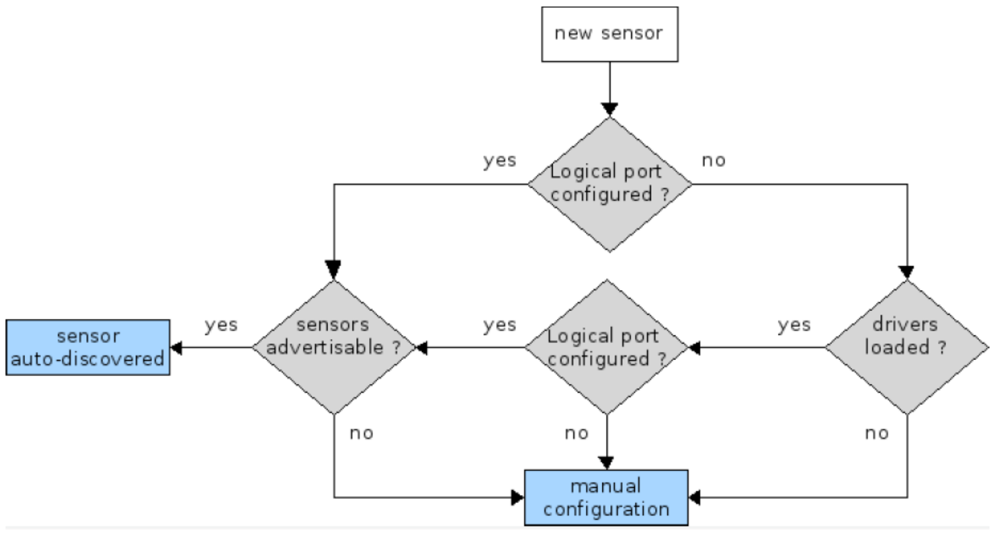

Fig. 4 Logic for New Sensor Discovery

To conclude, only some sensing technologies have support for full sensor auto-discovery, which is an advanced feature that is heavily hardware dependent. When a new sensor is connected to an existing logical port, if the native controller is unable to advertise it, then SAL must be configured manually with the new sensor information before it can be used. If the native controller is able to advertise the new sensor, then it can be detected and used straight away by SAL. Fig. 4 illustrates the logic for auto discovery of sensors. Further refinement of auto discovery remains the focus of future work.

V. Sensor Web Enablement and SAL XML standards

SAL abstracts the multiple heterogeneous sensor technologies and interfaces under a single generic API. To achieve this, there is a need for common ways of describing two important actors in the architecture – sensors and hardware platforms. Most of the time, users (and developers) have no knowledge of functions and features supported by various sensors and sensor gateways. SAL must have a standard way to convey information about a sensor's name, capabilities, supported commands and generated readings. In addition, the sensor gateway's hardware capabilities and current configuration also need to be specified in a standard format to facilitate automated configuration.

The mark-up languages described in this section fulfill this role: SALSensorML is used to describe sensors; CommandML is used for sensor command descriptions; and PCML describes capabilities and features supported by a platform, and its current configuration.

A. SALSensorML

The purpose of a SALSensorML document is to model processes by which sensors transform observable phenomena to data [1]. SAL uses the information contained in a SALSensorML document, for example, the valid range of readings to provide basic quality checks. Furthermore, SAL extends the OGC SWE SensorML standard with its own specific tags.

The goal of a SensorML document and its SAL extension is to provide SAL with the information required to:

1. Find a Sensor: All sensors can be referred to using their unique identifier. This allows sensors to be found anywhere in an architecture with multiple SAL agents, regardless of their location.
2. Address a Sensor: SAL must translate the sensor's unique identifier into the sensor’s native name to explicitly define the address. This is possible because a SensorML document provides a mechanism to match a unique identifier to a native name.
3. Send Commands to a Sensor: All commands supported by a sensor are listed in a CommandML section. Each command is listed along with a description of its input and output parameters.

SAL extends SensorML with the following:

- Unique Sensor Identifier: a SAL-generated string, which unambiguously identifies sensors. A three-level hierarchical namespace is used to ensure unicity.
- Native Sensor Addressing Information: SAL needs information on how to address a sensor in its own native naming scheme. The required information depends on the sensor. Some sensor technologies provide sensors with unique names (e.g., 1-wire and SNMP sensors for instance), while some others do not (such as LabJack sensors). Native addressing information are used for the following examples:

1. *SNMP Sensors*: SAL needs to know the SNMP agent (identified by its IP address), the object identifier of the element, and any other SNMP parameters required (such

as the community string, etc.).

2. *1-Wire Sensors*: SAL needs the 1-wire address of the sensor (and possibly the sensor on the 1-wire bus).
3. *LabJack Sensors*: SAL uses the LabJack connector name to refer to the connected sensor.

- Sensor commands: a list of commands supported by this sensor in CommandML.

B. CommandML

CommandML (CML) documents provide a common description, arguments and results of all the commands supported by a sensor and tie SAL generic commands to sensor-specific commands. Users who wish to pilot to find out about a particular sensor's capabilities typically retrieve the sensor's CML document. The document allows them to create semantically correct commands that can be sent to a SAL agent for execution. A CML document is technology-specific and is generated from a template by SAL's communication layer when a new sensor is added. The sensor can then be customised by the instrument owner.

C. PCML

Platform Capabilities and Configuration Markup Language (PCML) provides the language used to report and describe hardware platforms, which facilitates SAL's hardware management functions. SAL manages a wide variety of sensors so a common format to represent a platform's hardware and software capabilities is required. The purpose of a PCML document is to describe the hardware and software setup of a platform. The hardware capabilities are reported as a list of supported EndPoints and protocols. All possible configuration settings and associated values of the hardware are also listed. The hardware configuration document is similar to the previous one and reports a list of EndPoints and protocols, along with their current configuration settings.

D. Currently Supported Sensor Technologies

Table I lists the sensor technologies that are presently supported by SAL. That is, device drivers have been created that allow these sensors to work with the SAL system. Notably, this list will grow, as SAL is further developed to support new technologies based on the needs of the sensor network deployments that currently use SAL (described in Section VI).

VI. Real world SAL implementations

This section describes three real-world sensor network applications where SAL is currently deployed and is being used for further development of sensor network systems. These applications provide evidence for the versatility, suitability and cost-effectiveness of SAL in designing non-proprietary sensor networks. Table II provides an overview of the technical specifications for the three projects and a comparison of the costs.

TABLE I
SENSOR TECHNOLOGIES CURRENTLY SUPPORTED BY SAL

Instrument	Connection	Plug-in name	Sensor auto-detection	Adapter auto-detection
Operating system data	Local filesystem	OSData Protocol	Yes	Yes
SNMP devices	Ethernet	SSNMP Protocol	Yes	No
1-wire sensors	USB exp.	Owfs Protocol	Yes	Yes
V4L video sources	Any (relies on local filesystem)	V4L Protocol	Yes	Yes
Ambient Systems wireless sensor nodes	Serial	AS Protocol	Yes	No
Gumstix	USB / Serial		-	-
Java SunSPOT	USB	SALSpot	Yes	Yes
Odyssey	USB / Serial	Ody Protocol	Yes	Yes

TABLE II
TECHNICAL SPECIFICATIONS AND COST COMPARISON

Project	Hardware	Sensors	Scale	Cost
Davies Reef	Single Board Computer	Light, temperature	Small	$995
SEMAT	Gumstix Overo Air COM	Light, temperature, pressure, salinity	Medium – large	$4,000 per buoy
Smart Home	Seeeduino Stalker, personal computer	Temperature, humidity	Small - medium	$100

A. Davies Reef

A prototype of SAL was implemented and deployed on a single board computer to manage a network consisting of 30 sensors at Davies Reef [12]. Davies Reef is part of the Australia's Great. Barrier Reef located in North Queensland. The SAL prototype was used as a proof of concept, mainly to validate the network model used by SAL and the hardware abstraction. In this scenario, SAL managed sensors relying on four different technologies: 1-wire devices (temperature and humidity sensors), a serial device (battery charge controller), an SNMP device (microwave link modem) and pseudo sensors (OS status). The system cost $995 AUD.

B. Smart Environment Monitoring and Analysis Technologies (SEMAT)

The *Smart Environment Monitoring and Analysis Technologies* (SEMAT) project is driven by the need to create a low cost intelligent sensor system for undertaking environmental measurements and monitoring activities [13], [14]. SEMAT has been deployed for monitoring aquatic and coastal environments. The analysis of the collected data has been transformed into information that can be used for management and planning. The specific goals for SEMAT include: underwater wireless communications, short-range wireless power transmission, plug-and-play of sensor technologies, minimal deployment expertise, near real-time analysis tools and intelligent sensors (refer to [14] for further information).

SAL formed a core component of two SEMAT deployments that took place in Deception Bay and Heron Island in Australia during 2011 and 2012 respectively. During both deployments, five buoys were developed, each running a Gumstix Overo Air COM (Computer On Module). Odyssey data loggers where integrated into the system via the use of a SAL plug-in and seven data streams where recorded per buoy (temperature, light, pressure and salinity – at varying vertical profiles). A buoy cost approximately $4,000 Australian dollars to construct with sensors accounting for 45% of this total.

The two SEMAT deployments illustrate how SAL can be used successfully to prototype sensor network designs and create low cost sensor networks using heterogeneous technologies (i.e., different sensor types and hardware). Several future large-scale SEMAT deployments are planned for monitoring the health of waterways around resort islands in Fiji and Vietnam's Mekong Delta. For these deployments, SAL will be expanded with further sensor technologies of differing types, vendors, precision and cost.

C. Home Monitoring

SAL has also been trialled in a low-cost and easy-to-maintain environmental monitoring WSN that is suitable for deployment into a home environment. The system utilised a homeowner's existing personal computer and Internet connection to collect temperature data from sensors attached to an Arduino-based platform. SAL was ported to the Windows operating system to enhance the end-user experience. Data was automatically transferred from the sensing platform to a central database whenever a network connection to the personal computer was detected. The sensors used were custom-build DS18B20 digital temperature sensors (worth less than $2). The cost for a basic sensor network is less than $100. Such a system has important ramifications for researching more effective use of electricity in the home.

VII. Conclusions

This paper presented SAL – a middleware integration platform for sensor networks. SAL aggregates multiple sensor networks and provides a generic, hardware-independent interface to manage and control sensors. SAL is unique in that there are no existing solutions that offer this approach for designing heterogeneous sensor networks that can use technologies from multiple vendors. This approach offers the following benefits:

- A flexible interface to integrate disparate sensor technologies.
- The removal of compatibility obstacles that hamper the addition of sensors that are best suited (and priced) for a specific task.
- A lightweight system for devices with limited processing power, battery life, and storage capacity.
- New technologies can be quickly incorporated with minimal amount of changes to software code.
- A SAL-managed sensor can be a single transistor-like device, a discrete wireless node or a part of a larger instrument.
- Sensors are automatically detected and configured where supported by the technology.
- Forward compatibility due to adherence to the SWE standards proposed for sensor network systems.
- Remote operation of the sensor network and near real-time streaming of the data collected.

Prototype systems have been successfully deployed at Davies Reef and as part of the SEMAT initiative at Deception Bay and Heron Island (Australia). SAL has also been trialled as part of a "smart home" initiative to monitor parameters to design more energy efficient households. These projects illustrate SAL's versatility for differing applications and how SAL enables the construction of inexpensive heterogeneous systems. SEMAT's success will potentially make SAL a standard lower middleware solution for many upcoming marine sensor network deployments around South East Asia and the Pacific region.

Further work involves improving the user interface for the SAL client and moving towards stricter adherence with SWE Standards. Furthermore, as SAL continues to evolve, more software plug-ins will be developed that will increase the number of sensor devices that are compatible with SAL.

Acknowledgment

The authors would like to acknowledge the contribution of Gilles Gigan for the initial development of the SAL concept and Professor Wayne Read for his advice and expertise. We would also like to thank all the members of the SEMAT team.

References

[1] G. Aloisio, D. Conte, C. Elefante, G. P. Marra, G. Mastrantonio, and G. Quarta, "Globus Monitoring and Discovery Service and SensorML for Grid Sensor Networks," in Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Manchester, United Kingdom 2006, pp. 201-206.

[2] R. Bramley, K. Chiu, T. Devadithya, N. Gupta, C. Hart, J. C. Huffman, K. Huffman, Y. Ma, and D. F. McMullen, "Instrument monitoring, data sharing, and archiving using common instrument middleware architecture (CIMA)," Journal of Chemical Information and Modeling, vol. 46, pp. 1017-1025, 2006.

[3] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," in Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking (MobiCom '99), Seattle, WA, USA, 1999, pp. 174-185.

[4] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, "Middleware to support sensor network applications," Network, IEEE, vol. 18, pp. 6-14, 2004.

[5] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," in Proceedings of the 10th international conference on Architectural support for programming languages and operating systems, San Jose, California, USA, 2002.

[6] T. Liu and M. Martonosi, "Impala: a middleware system for managing autonomic, parallel sensor systems," in Proceedings of the ninth ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP 2003), San Diego, California, USA, 2003, pp. 107-118.

[7] K. Römer, O. Kasten, and F. Mattern, "Middleware challenges for wireless sensor networks," ACM SIGMOBILE Mobile Computing and Communications Review, vol. 6, pp. 59-61, 2002.

[8] E. Souto, G. Germano, G. Vasconcelos, M. Vieira, R. Nelson, C. Ferraz, and J. Kelner, "Mires: a publish/subscribe middleware for sensor networks," Personal and Ubiquitous Computing, vol. 10, pp. 37-44, 2006.

[9] V. Venkatesh, P. Raj, and V. Vaithayanathan, "A Device Middleware-Based Smart Home Environment for Ambient Health Care," Global Trends in Computing and Communication Systems, vol. 269, pp. 144-153, 2012.

[10] M. M. Wang, J. N. Cao, J. Li, and S. K. Dasi, "Middleware for wireless sensor networks: A survey," Journal of computer science and technology, vol. 23, pp. 305-326, 2008.

[11] T. Hasiotis, G. Alyfantis, V. Tsetsos, O. Sekkas, and S. Hadjiefthymiades, "Sensation: a middleware integration platform for pervasive applications in wireless sensor networks," in Proceedings of the 2nd European Workshop on Wireless Sensor Networks., Istanbul, Turkey, 2006, pp. 366-377.

[12] C. Huddlestone-Holmes, G. Gigan, and I. Atkinson, "Infrastructure for a sensor network on Davies Reef, Great Barrier Reef," in 3rd International Conference Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 07), Melbourne, Australia, 2007, pp. 675-679.

[13] R. Johnstone, D. Caputo, U. Cella, A. Gandelli, C. Alippi, F. Grimaccia, N. Haritos, and R. E. Zich, "Smart Environmental Measurement & Analysis Technologies (SEMAT): Wireless sensor networks in the marine environment," in ICT-Mobile Summit, Stockholm, Sweden, 2008.

[14] J. Trevathan, R. Johnstone, T. Chiffings, I. Atkinson, N. Bergmann, W. Read, S. Theiss, T. Myers, and T. Stevens, "SEMAT – The next generation of inexpensive marine environmental monitoring and measurement systems," Sensors, vol. 12, pp. 9711-9748, 2012.

[15] P. Bonnet, J. Gehrke, and P. Seshadri, "Querying the physical world," Personal Communications, IEEE, vol. 7, pp. 10-15, 2000.

[16] V. Handziski, J. Polastre, J. H. Hauer, C. Sharp, A. Wolisz, and D. Culler, "Flexible hardware abstraction for wireless sensor networks," in Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on, Istanbul, Turkey, 2005, pp. 145-157.

[17] D. Le-Phuoc, H. N. M. Quoc, J. X. Parreira, and M. Hauswirth, "The Linked Sensor Middleware: Connecting the real world and the Semantic Web," in Presented in the Semantic Web Challenge 2011, 10th International Semantic Web Conference (ISWC 2011), Bonn, Germany, 2011.

[18] R. Beaubrun, J.-F. Llano-Ruiz, and A. Quintero, "An Approach for Designing and Implementing Middleware in Wireless Sensor Networks," Sensors and Transducers Journal, vol. 14-2, pp. 150-163, 2012.

[19] M. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC® Sensor Web Enablement: Overview and High Level Architecture," in GeoSensor Networks. vol. 4540/2008 Berlin: Springer Berlin / Heidelberg, 2008, pp. 175-190.