PRISMA: A Publish-Subscribe and Resource-Oriented Middleware for Wireless Sensor Networks

José R. Silva, Flávia C. Delicato, Luci Pirmez, Paulo F. Pires, Jesus M. T. Portocarrero
PPGI-DCC/IM
Federal University of Rio de Janeiro, UFRJ
Rio de Janeiro, Brazil
{jr.joserenato.jr, fdelicato , luci.pirmez, paulo.f.pires, jesus140}@gmail.com

Taniro C. Rodrigues, Thais V. Batista
DIMAp
Federal University of Rio Grande do Norte, UFRN
Natal, Brazil
{tanirocr, thaisbatista}@gmail.com

***Abstract*—PRISMA is a resource-oriented publish/subscribe middleware for WSN, which the main goals are to provide: (i) programming abstraction through the use of REpresentational State Transfer (REST) interfaces, (ii) services, encompassing asynchronous communication, resource discovery and topology control, (iii) runtime support through the creation, configuration, and execution of new applications in WSN, and (iv) QoS mechanisms to meet applications constraints. This paper describes PRISMA architecture, its implementation in the Arduino platform, and a preliminary evaluation.**

Keywords - middleware; publish/subscribe; topology control.

I. INTRODUCTION

Wireless Sensor Network (WSN) technology has been evolving fast in recent years. There are currently several hardware platforms available for WSN, such as the sensor motes manufactured by MEMSIC (former Crossbow [1]), Sun Spots [2] (now Oracle) and, more recently, Arduino platform [3], that is used in this work. Additionally, WSNs have gained a lot of attention in the research community and are becoming increasingly popular in the industry, due to their wide range of potential applications.

Early applications developed for WSN presented simple requirements and did not demand the use of complex software infrastructures. Moreover, WSN were typically designed to meet the requirements of a single target application. In other words, the source code installed in the nodes was commonly monolithic, highly tied to the requirements of a single application and to a specific sensor platform and the protocol stack for such platform. Furthermore, the application development was highly coupled to low-level primitives provided by the WSN operational system and the design approach was focused on improving the network energy efficiency, given the limited resources of nodes. Such dependence between the application layer and the underlying layers (protocols and hardware) is not desirable for emergent applications and new trends in the field, where the same physical infrastructure of a potentially heterogeneous WSN may be used for various applications, whose requirements are not known at the network deployment time [4]. As the number of WSN physical infrastructure currently deployed is increasing, there is a trend to share and integrate the sensing data produced by these networks through different applications, as well as growing initiatives to include monitoring data as part of Web applications, integrated to other types of resources available on the Internet. In such scenario, there must be interoperability between different WSNs, possibly between different applications, and between WSNs and external networks, as the Internet.

In order to meet the emerging trends of WSN scenarios, there is a need to adopt software platforms at the middleware level. A middleware can provide abstractions to build applications and to access data produced by the network, and offer generic or domain specific services. It can also provide a uniform API and standardized protocols that allow interoperability in an environment with high degree of heterogeneity. Despite of the fact that middleware platforms are widely used in traditional distributed systems, their development in the WSN context is relatively recent [4].

A WSN middleware is a layered software that lies between application code and the communication infrastructure providing, via component interfaces, a set of services that may be configured to facilitate the application development and execution in an efficient way for a distributed environment [5]. Thus, the main goal of a middleware is to enable the interaction and the communication between distributed components, hiding from application developers the complexity of the underlying hardware and network platforms, and freeing them from explicit manipulation of protocols and infrastructure services. Besides these generic requirements, a WSN middleware needs to consider some basic features, specific to this context. According to Wang et al. [4], a WSN middleware should offer four main features: (i) programming abstractions, (ii) services, (iii) runtime support, and (iv) mechanisms for Quality of Service (QoS) provision. Programming abstractions define the interface of the middleware for the application developer. Services provide implementations to achieve the abstractions; thus, services encompass the functionalities provided by the middleware and comprise the middleware core. Runtime support acts like an extension of the embedded operating system to support the middleware services. Finally, QoS mechanisms are used to meet quality constraints imposed by applications such as network lifetime, coverage, accuracy, latency, bandwidth,

 ISBN: 978-1-61208-360-5

and others. There are several works [6]–[9] proposing middleware platforms for WSN addressing issues such as interoperability between heterogeneous devices, support for multiple application domains, adaptation and context awareness, service discovery, management of devices and other features. However, few of these works address all four requirements of WSN middleware aforementioned [4].

In this context, this paper introduces PRISMA, a resource-oriented publish/subscribe middleware for WSN, which aims to provide the aforementioned main functionalities required for WSN middleware. PRISMA programming abstraction for client applications is based on REpresentational State Transfer (REST) [10]. REST defines a lightweight communication between applications based on Web standards to facilitate the access to sensor generated data. Using a REST-based approach, the WSN, its nodes, and the sensing units of each node are described and accessed by end users and client applications as *resources*, in the same way as traditional Web resources are accessed through the Internet. By providing a high level and standardized interface for data access, PRISMA allows interoperability of networks from different technologies, thus aligning to the current trend of building heterogeneous systems involving multiple networks and applications.

PRISMA functionalities (see Section II) include (i) mechanisms to facilitate the creation and execution of WSN applications; (ii) a topology control service aiming at efficiently managing the energy consumption of nodes; (iii) capability of configuring applications QoS parameters, such as network lifetime and maximum delay; (iv) asynchronous communication via the publish/subscribe paradigm. This last is a significant feature since several WSN applications are event-driven; thus, the traditional request-reply communication model is not proper for most scenarios.

Although its logic architecture is agnostic regarding the underlying sensor platform, PRISMA physical design and implementation were tailored to Arduino-based platforms. The main motivation for using Arduino is the fact it is open-hardware, still poorly explored by the academic community of WSN, mainly in the middleware field. Moreover, this platform provides a high-level language that can be leveraged to facilitate the application development.

The rest of this paper is structured as follows: Section II presents PRISMA specification and logic architecture; Section III describes the implementation for Arduino platform; Section IV discusses related work; Section V presents performed evaluations, and finally, Section VI contains conclusions and future work.

II. PRISMA

This section presents an overview of PRISMA, its logic and physical architecture, the system operation, and the available services.

A. OVERVIEW

PRISMA assumes a heterogeneous and hierarchical WSN, with three levels: (i) Gateway, (ii) Cluster Head, and (iii) Sensor Node. The top level is represented by Gateways that are responsible for managing the network from a high level viewpoint, taking the global decisions on the system operation. In the intermediate level, Cluster Heads locally manage their respective clusters in the network. Each cluster encompasses a set of sensor nodes and one cluster leader (the Cluster Head). Cluster Heads require higher computational power than ordinary nodes since they are in charge of managing functions for a part of the network. This additional computational power is required to store information about the nodes in each cluster. Finally, in the lower layer a huge number of sensor nodes (also called as ordinary nodes) are responsible for collecting environmental data and taking local decisions. This hierarchical approach was adopted to promote scalability and facilitate network management.

PRISMA adopts REST design pattern to facilitate the access to WSN data and to support interoperability with other networks. PRISMA communication service encompasses the communication with the WSN nodes and between the middleware and external networks (as the Internet). The communication is provided by a Web Server and a broker to provide asynchronous communication. In the development of the middleware communication service, we adopted REST to communicate with client applications. In REST-based Web services, the uniform interface for accessing resources is given by Hypertext Transfer Protocol (HTTP) methods. The middleware designer has the responsibility of defining the granularity of the provided REST resource: (i) the entire WSN can be seen as a unique resource; (ii) each individual node can be exposed as a resource; and (iii) there may be many resources in each node, for example, each sensing capability (temperature, light, etc.) deployed in one single sensor node.

Besides using REST interfaces to interact with client applications, PRISMA adopts the publish-subscribe paradigm to notify its clients about events of interest. To receive notification messages, a client application must be subscribed in a publish-subscribe topic. A publish-subscribe topic is an asynchronous communication channel used by the middleware to publish interest messages to client application. This topic will be created if the application requirements can be satisfied with the WSN resources. If the WSN can meet the requirements, the client will receive the topic in response to the REST request; otherwise, will receive an error message. With this response information the client will subscribe to the desired topic in PRISMA broker and receive the data of interest whenever it is available.

B. ARCHITECTURE

PRISMA design follows a layered architecture, shown in Fig. 1, composed of three layers (i) Access, (ii) Service, and (iii) Application, described as follows. A brief description of

 ISBN: 978-1-61208-360-5

the services provided by PRISMA software components will be presented after the description of the three layers.

- Access layer: consists of four components: **Communication, Data Acquisition, Context Monitor and Topology control.** The **Communication component** is responsible for receiving and extracting data from messages transmitted by the WSN nodes. This component includes drivers for translation and composition of messages that travel in the WSN and a listener to capture messages. The **Data Acquisition component** manages the data collection via sensing units of the nodes. The **Context Monitor component** is in charge of monitoring the network execution context in order to verify if QoS requirements are being fulfilled. An example of monitored context is the energy level of devices, which directly relates to the QoS requirement of network lifetime. The **Topology control component** is responsible for the network logical organization. This component performs the initial network configuration and a reconfiguration whenever (ii) a new application arrives, (ii) the network energy level is lower than a critical parameter, or (iii) a device failure occurs.
- Service layer: consists of three components: **Event,** responsible for managing and notifying requested events from applications in execution; **Publish and Discovery,** responsible for registering and publishing new services to be offered by the network (providing PRISMA resource discovery service); and **Decision,** responsible for analyzing arriving applications in order to verify available devices that satisfy the specified applications requirements. **Decision** is the decision-making center of the middleware (further described later).
- Application layer: consists of two components: **Application Control, Publish and Subscribe Proxy** and the **Web Server.** The first is responsible for receiving and managing applications sent to the WSN through a REST interface in a configuration file. Upon a parse of the file, its content is forward to the **Decision** component. The **Proxy Publish and Subscribe** allows asynchronous communication with client applications through the publish-subscribe paradigm. This component acts as a broker that manages the queues of PRISMA publish-subscribe implementation [11]. The **Web Server** is responsible for providing the REST interfaces that PRISMA offers, such as: (i) *Create* interface that receives new applications to be executed on the WSN; (ii) *GetServices* interface responsible for advertising the services available in the WSN; (iii) *GetData* interface, responsible for querying the data collected by the WSN (historical or current data).

These software components are divided into three subsystems, each one corresponding to a different level of the physical components considered in our architecture: (i) **Gateway** (ii) **Cluster Head** and (iii) **Sensor Nodes**. At the Gateway subsystem all components of the architecture are deployed, except the Data Acquisition component. At the Cluster Head subsystem all components of the Service Layer and Access Layer are deployed, except the Data Acquisition component that is specific to the Sensor Node subsystem. At the Sensor Node subsystem all components of the Service Layer and Access Layer are deployed. The only subsystem that communicates with client applications is the gateway subsystem for being the one that includes all the components of the Application Layer.

PRISMA provides four services: (i) communication; (ii) topology control; (iii) resource discovery; and (iv) context monitoring. The communication service is responsible for the communication among middleware components and the WSN nodes, and with external entities (client applications or the Internet). This service is provided by the following

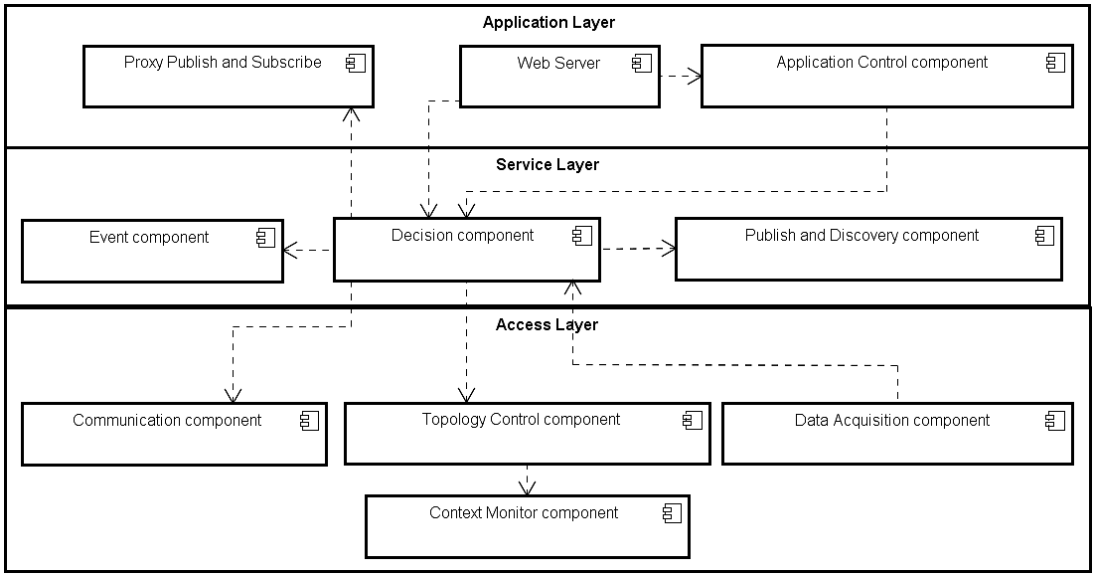

Figure 1. UML component diagram illustrating the PRISMA architecture

components: communication component to exchange messages with the WSN; Web Server to communicate with external networks and Proxy Publish and Subscribe to communicate asynchronously with client applications. The topology control service is responsible for selecting the clusters and nodes that will participate in the sensing tasks for a given application; this service is provided by the topology control component. The resource discovery service is provided by the Publish and Discovery component. This service is responsible for identifying new nodes in the network and publishing new available services to the decision maker center of the middleware. The context monitor service is provided by the Context Monitor component and it is responsible for monitoring the energy of WSN/Cluster/Node (depending on the subsystem) and detecting conditions of lacking of energy.

C. System Operation

This subsection presents PRISMA operation from the sensor nodes deployment to the creation and configuration of applications on WSN nodes. The UML diagram activity of Figure 2 depicts the main steps of this operation and following we briefly describe these steps.

Initially, the middleware (software) components are installed on physical devices according to their functionality (**Cluster Heads** or **Sensor Nodes**). Information about the geographical area of deployement and the Cluster Head assigned for each node are "hard-coded" into the code of the nodes. Then, considering that the nodes are distributed in their respective target areas, the **Resource Discovery** service starts. This service is responsible for identifying each sensor node that is active in a specific geographic area as well as its sensing capabilities (the provided services/resources). The process starts with the sensor node sending a message to their respective Cluster Head. Then, the **Cluster Head** updates its node list and sends a message to the Gateway containing the description of the set of sensing capabilities managed by the (new) nodes to the decision-making center of the middleware. This message includes, for each sensor node, the following information:

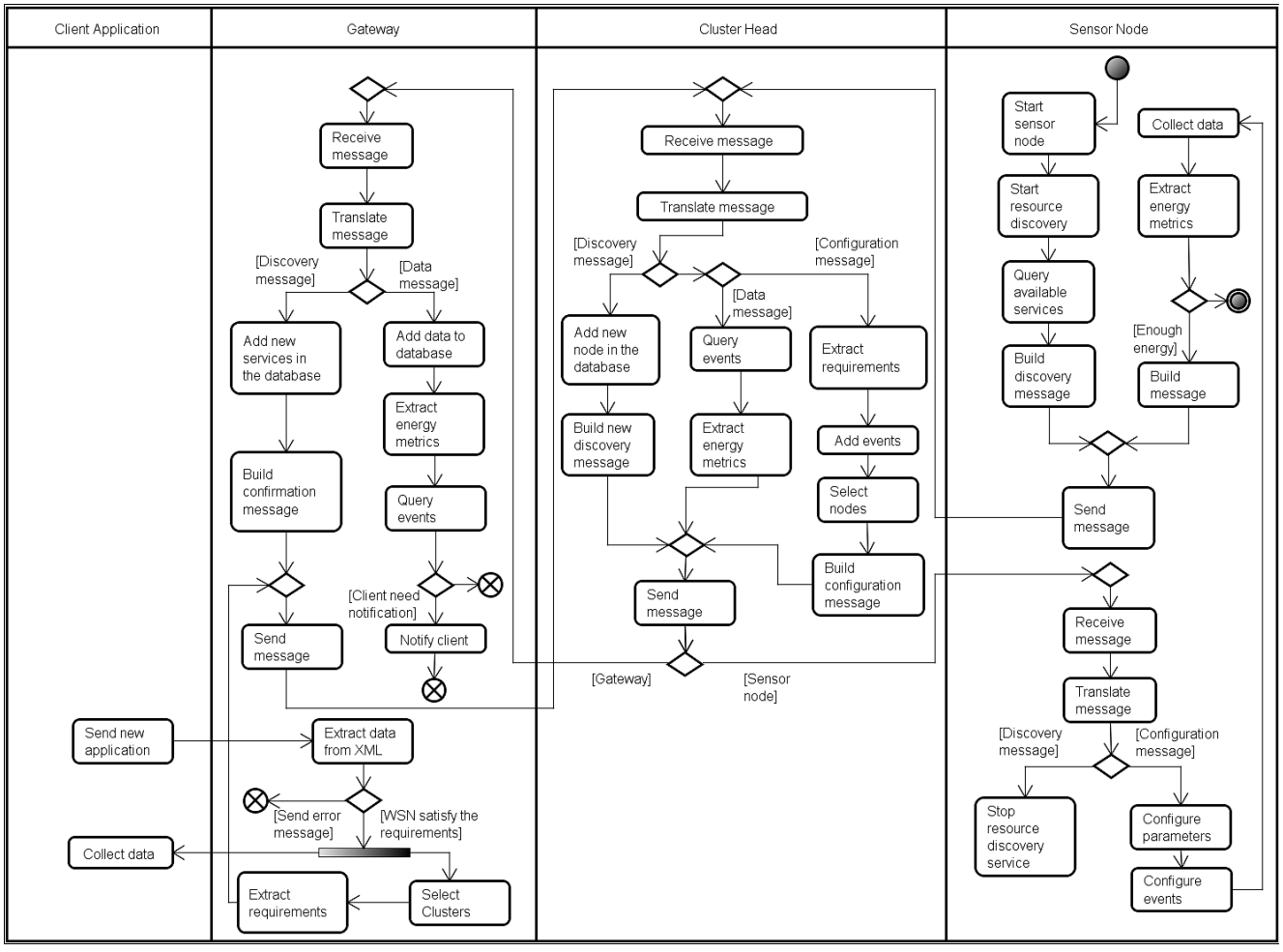

Figure 2. UML Activity diagram of PRIMA operation

(i) address defined in the sensor node radio (MAC address), (ii) cluster to which it belongs (Cluster ID), (iii) available resources (sensing units, as for example, temperature, humidity) and (iv) residual energy. The **Gateway** then updates its database with this information using the **Publication and Discovery** component. The **Context Monitor** is responsible to identify when a node's energy is almost depleted and notify the Resource Discovery service to advertise the Cluster Head and Gateway of its low energy in the same process described above.

Once the network is organized, all its resources are listed and made available through the **Gateway**, applications can be created in the WSN. The Gateway receives new applications to be deployed in WSN through a REST interface. Hence, in PRISMA all access to the WSN resources and creation of new applications follows a RESTful approach. Configuration files are submitted via a REST interface to create applications and each file is translated in parameters to configure the network. These parameters are sent to specific clusters, which are responsible for organizing themselves in order to provide required data and services. This organization comprises the selection of the nodes to actively participate in data collection following the requirements sent by the application. Applications can be created through the *Create* REST interface (one of the REST interfaces available on PRISMA by the **Web Server** component). Such interface receives an eXtensible Markup Language (XML) [12] file through a HTTP POST message [13], in the following url: http://ServerAddress:8080/prisma/rest/applications/create. Figure 3 depicts an example of a configuration file.

The process to create a new application starts by receiving a XML configuration file (Figure 3) that specifies the application requirements to execute into the nodes (sent by the client application); this XML file is translated in one Java object by the **Application Control** component and, after that, these requirements are verified to define whether the required services (specific ability of every sensor node) may be attended by one or a set of available sensor nodes, the **Decision** component query the Publish and Discovery component to check the available services to verify this. After that, requirements are translated to parameters to be configured in the selected nodes to meet the specified requirements. A configuration file may have many services where each one defines a sensing task. A XML configuration file defines: (i) a periodic application, for instance, to monitor temperature of an environment every 5 minutes; (ii) an event-driven application, for instance, to monitor temperature in an environment and to notify the client whenever a sensor detects a value of 50ºC or more; or (iii) both types of applications, for instance, to monitor temperature of an environment every 5 minutes and to notify when a temperature achieve 50ºC in order to detect fire. PRISMA middleware supports multiple applications by reusing WSN data or allocating new nodes to be active.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Application>
3      <username>username</username>
4      <collectionRate>1000</collectionRate>
5      <maxDelay>1000</maxDelay>
6      <lifetime>15d</lifetime>
7      <services>
8          <sensorType>Humidity</sensorType>
9          <targetArea>Lab. 1</targetArea>
10     </services>
11     <events>
12         <service>
13             <sensorType>Temperature</sensorType>
14             <targetArea>Lab. 2</targetArea>
15         </service>
16         <superiorLimit>35</superiorLimit>
17         <inferiorLimit>35</inferiorLimit>
18         <operator>">"</operator>
19         <targetArea>Lab. 2</targetArea>
20     </events>
21 </Application>
```

Figure 3. New application configuration file

Figure 3 describes an application ready to be executed in the network. It is possible to define the following requirements for an application: (i) data collection rate (in milliseconds), (ii) maximum delay (in milliseconds), and (iii) the application lifetime, which can be set in hours or days. In addition to these requirements, this configuration file also defines which services are required and in which geographical area these services should be located. The target areas are statically configured (at pre-deployment time) by the WSN administrator in the sensor nodes. Between lines 7 and 10 of the example we define the service to monitor **humidity** data in the area. This data should be collected from the **Lab. 1** (a symbolic region) and respect the collection rate set in line 4 (1 second).

We can also define events that will be monitored by setting the service, target geographic area, upper/lower limit when applicable and the comparison operator being used. The **Event** component is responsible for recording these events and checks them each time new data is received. Lines 11 to 20 specify to collect **temperature** data and notify (send data messages) only when they are higher than 35 degrees in the geographical area named **Lab 2**. The publish-subscribe topic for this application is created by the **Proxy Publish and Subscribe** after receiving this configuration file. The access information to the topic (the topic name) is sent as a response through the REST interface accessed by the client. The client subscribes this topic to receive the asynchronously requested data.

The **Decision** component is responsible for analyzing the arriving sensing applications, extract its requirements and query the **Topology Control** component to verify the existence in the WSN of devices (nodes) that meet the requirements specified in the received configuration file. If any device meets the requirements specified, the **Decision** component will create a message and the **Communication** component will send the application requirements to the Cluster Head(s) of the respective devices. Only clusters that are selected for the execution of the application will receive

 ISBN: 978-1-61208-360-5

these requirements. Each cluster is only assigned with tasks that can be met by its currents resources, thus avoiding sending unnecessary messages. In PRISMA, a task corresponds to the act of executing a service (for instance, a temperature sensing task) and a service corresponds to a given capability of a node (for instance, the capability of sensing temperature values).

When the requirements are received in the **Cluster Head** the local process of selecting active nodes is started (more details in Section D). After the **Topology Control** component selects the nodes that will participate in the data acquisition for the application, the received task requirements are then translated into parameters by the **Decision** component to be configured and sent to the selected nodes. These parameters are directly extracted from the configuration file: sensing capabilities required, collection rate, maximum delay, application lifetime and threshold for sending data (representing the detection of an event of interest). The selection of active nodes takes into account the residual energy of nodes, information that is updated whenever a message is sent by the node.

D. Topology Control

A major goal in the management of nodes in WSNs is to rationalize the use of energy resources of the network in order to prolong its lifetime and consume energy evenly across the nodes. One way to achieve this goal is by adopting a scheme of rotation of the work performed by the network nodes, changing their operating mode (active or sleep mode), where the subset selected to remain active should be able to meet the requirements requested by the application.

The problem of selecting active nodes can be expressed as the algorithm that decides which sensors must remain active for a given application task. In PRISMA, this is the responsibility of the topology control algorithm, the core of the middleware topology control component. The algorithm proposed in this paper is based on [14], where the time is divided into *j* rounds during which the selected subset of nodes remains constant. A task starts running at the beginning of a round and can last for a time equal to an integer multiple of *p*, where *p* is the length of a round.

The mechanism of topology control is first executed when the requirements of an application are sent to the network. These requirements contain a description of the application that will run on the WSN and the desired QoS requirements. After the execution of the topology control mechanism the first round for the application in question starts. The selection algorithm can be run again in the following cases: (i) on demand by the application to change any parameters of QoS if needed, (ii) in a proactive way by the network, for the purpose of energy conservation, for instance, and (iii) reactively by the network, when the **Context Monitor** component detects a QoS requirement is not being met.

Unlike the proposal of the algorithm described by Delicato et al. [14], that is based on a flat network of homogeneous sensors, and where the process of selecting active nodes is centralized, in PRISMA the network is heterogeneous, and a hierarchical selection is performed in two levels: the first level corresponds to the global view of the network and second level corresponds to the local view of the network, within each cluster. PRISMA approach has the potential of allowing greater scalability since it is performed at two levels, thus being more suitable for scenarios of large-scale and shared WSNs. The algorithm does not need to flood the network to determine active nodes for every application to be executed.

By adopting a hierarchical approach, in PRISMA the topology control mechanism runs on two physical components: (i) **Gateway,** and (ii) **Cluster Head**, where each component performs the algorithm on a different level. The first level corresponds to the global view of the network (**Topology Control Component** of the **Gateway**) where the **Gateway** is responsible for a pre-selection of clusters that contain sensor nodes potentially useful for an application. At this level, the **Gateway** runs the steps of the topology control algorithm to determining the clusters to be used for a given application: clusters that do not have resources or capabilities to suit a given application will be excluded from active nodes selection process. The exclusion of these clusters is based on a simple set of rules, for example, exclusion of clusters not having the necessary services or those outside the desired geographical area. The second level, local within each cluster, will run in the **Cluster Heads** that have a higher processing power than the ordinary sensor nodes. The higher processing power and memory capacity is desired to maintain in the Cluster Head the list of nodes that are in its coverage area and information about these nodes, such as available services, energy level and its state (sleep or active). The process of changing the operating mode of the node is implemented through configuration/control messages responsible to set the duty cycle of each node.

After the pre-selection executed by the Gateway, the algorithm for selecting active nodes is triggered in a Cluster Head (selected in the first level of the topology control mechanism) whenever it receives a request to create a new application, or when the energy level of any node is below a minimum threshold for the execution of their tasks. Such algorithm running at the cluster heads receives as input the application requirements (data from the XML file) and the set of available services on its cluster, and produces as output the set of nodes to be used by the application.

The process of selecting active nodes begins with a query to retrieve the energy levels of the cluster nodes, maintained by the **Cluster Head** responsible for a given area. After running the algorithm for the selection of active nodes a message is sent to each node in the cluster to determine whether the node is active or in sleep mode. If it is in sleep mode, the next time the node wake up it will

 ISBN: 978-1-61208-360-5

query its cluster head to check pending messages and then receive a control message. Information about energy stats is collected and sent along with the data collected by the sensors in order to supply its Cluster Head with the information on current energy levels of the network.

III. IMPLEMENTATION

The components of the **Gateway** subsystem were developed on J2EE 1.4 platform and implemented using: Apache TomEE [15] as application server, Jersey for the creation of REST interfaces; Log4J for handling logs (for debugging purposes), Hibernate to implement the persistency layer, and MySQL relational database management system [16] as the data repository. The project was developed following the design pattern Data Access Object (DAO) [17]. Asynchronous communication was developed using ActiveMQ [18] which is included in Apache TomEE. The source code can be found in the URL: http://ubicomp.nce.ufrj.br/ubicomp/projetos/prisma/.

As stated in Section I, the target sensor platform for PRISMA implementation is Arduino. The main motivation for this choice is that this is a recent platform (launched in 2005), open hardware, not explored by the academic community of sensor networks, especially in the area of middleware. To the best of our knowledge, there is currently no WSN middleware implementation in this platform reported in the literature up to date. Furthermore, the platform has a high level language which facilitates development of applications.

TABLE 1. COMPARISON TABLE OF ARDUINO MODELS

	Arduino UNO	Arduino MEGA
Microcontroller	ATmega328	ATmega1280
Operating Voltage	5V	5V
Recommended input voltage	7-12V	7-12V
Input voltage limit	6-20V	6-20V
Digital input and output pins	14 (6 can provide power)	54 (15 can provide power)
Analog input pins	6	16
Current output for I / O pins	40mA	40mA
Pin 3.3V current output	50mA	50mA
Flash memory	32KB (0.5KB is used by the bootloader)	128KB (4KB are used by the bootloader)
SRAM	2KB	8KB
EEPROM	1KB	4KB
Clock speed	16MHz	16MHz

The components of the **Cluster head** and **Sensor node** subsystems were developed using the Arduino IDE development that is available at Arduino official web site. The nodes were programmed in Arduino Programming Language [19] (based on Wiring programming language [20]). This language has three main categories of code constructs: structures, values (variables and constants) and functions. Such a language is based on C / C++ [21]. Given this fact, any function of these languages can be used in Arduino programming. As previously mentioned, PRISMA works with a heterogeneous network, where the Cluster Heads need more computing power. Therefore, Arduino MEGA was chosen for this function since it has higher computational power. The sensor nodes use the Arduino UNO model. Additional hardware details of these models can be found in Table 1. The platform offers the concept of shields, which are cards that can be added to the Arduino board to increase its functionality. There are Arduino shields for connecting with Bluetooth, Ethernet modules, among others. The shield used in this work, called XBee Shield allows the interconnection of the Arduino XBee radio module [22].

PRISMA has a set of libraries representing the following features of the middleware: topology control, services discovery, and a library for message handling. In addition to the libraries specifically developed to implement PRISMA, the following existing libraries are used: XBee-Arduino [23], responsible for communicating with the XBEE radio and PString [24] to facilitate the use of the API functions and so reduce the complexity of handling messages.

XBee radio works with two operating modes for data transmission and reception. In the first mode, called Transparent Operation or **AT**, the data is sent and received directly through the node serial port. In order to send data and use AT commands, the application code installed on nodes needs to connect to the serial port of the XBee module. Through AT commands it is possible to modify the XBee configuration of the node. This mode although simple is not scalable to send data to multiple recipients and in order to change the XBee radio configuration it is necessary to access to the physical device directly.

The second mode, which is used in this work, is the Application Programming Interface (API) mode, based on sending and receiving data frames by specifying how commands, command responses and messages about the operating status of the XBee module are sent and received. This mode allows remotely sending settings (AT commands) for the XBee radio of the nodes. By using the API mode, new nodes can be inserted in the WSN and configured on the fly, thus facilitating scale up the network. AT commands can also be sent and received via the API mode, allowing the coexistence of the two modes in one network. By using the Arduino in conjunction with the XBee radio for wireless communication it is possible to encapsulate the data exchanged over the network in packets that follow the IEEE 802.15.4 standard [25].

IV. RELATED WORK

This section analyzes existing publish-subscribe middleware platforms for WSN and compares them with PRISMA. In [26], TinyDDS is described as a (re)configurable and open source middleware, developed using design patterns, and aimed at offering interoperability of publish-subscribe WSN applications. TinyDDS addresses

 ISBN: 978-1-61208-360-5

most of the features mentioned in Section I, except the QoS mechanism. Moreover, TinyDDS does not consider adaptation issues and does not include a topology control mechanism. In this sense, PRISMA provides is a more comprehensive middleware solution.

MiSense [27] is a service-oriented and component-based middleware designed to support distributed applications running in sensors with different performance requirements. MiSense covers all the features described in Section I, but the services offered by MiSense are simple, e.g., (i) the topology control mechanism merely elects cluster heads based on their energy levels and makes these nodes responsible for forwarding all messages originated within that cluster to the sink node, overloading the cluster heads and depleting the energy quickly in a high workload condition. On the other hand, the algorithm used in PRISMA considers the application requirements to build the WSN logical topology, thus its solution tries to balance between optimization of the networks resources and the needs of final users. Also, in [27] (ii) there is an asynchronous communication service where each node has its own Broker that manages the topics and subscriptions. However, this approach based on Broker can overload nodes with high workloads because each node consumes most of its available battery transmitting the new collected data for all subscribed applications/nodes. In PRISMA, the Gateway takes those responsibilities and works as an intermediary between applications and sensor networks, avoiding excessive exchange of messages by the use of the publish-subscribe mechanism.

MufFIN [9] (*Middleware For the Internet of thiNgs*) is a IoT middleware that uses SOA principles and Sensor Web Enablement (SWE) to provide an abstraction layer to client applications. The main contributions of MufFIN are: (i) providing programming abstraction to applications and (ii) management of collected data and its broadcast to applications via Web services respecting the SWE specifications. MufFIN provides abstraction of code deployment, communications and hardware of smart objects. The authors have chosen to accommodate the differences between the heterogeneous devices at the middleware level. From the perspective of the application, all devices are reprogrammable and can communicate. MufFIN allows all its connected devices to be reprogrammable even though the device does not have this capability natively. In order to enable this feature, the middleware creates a filter (called Data-Flow) to process the information received from the WSN. For the application such approach works as if the device is running the code, but in fact the data collected pass through the Data-Flow provided by the middleware and only after such process it is delivered to the client. Differently from PRISMA, MufFIN does not provide any support for QoS management. By encompassing a topology service that also works as a QoS mechanism, PRISMA aims at providing a more complete solution, at the middleware level, for WSNs.

Mires [28] is a middleware based on both service and publish-subscribe paradigms that operates above the TinyOS layer encapsulating its interfaces and providing high-level services to applications. Internally, Mires consists of a publish-subscribe service, a routing component and additional services. Although Mires adopts a service-based design and provides asynchronous communication, it does not offer programming abstraction, runtime support or QoS mechanisms. The only mechanism to save energy included in Mires consists in reducing the number of messages sent in the network. This is accomplished by sending only messages related to subscribed topics. In PRISMA the energy is saved by the topology control algorithm. In PRISMA algorithm the requirements of client applications are checked and only target clusters and nodes within the interest area and able to provide useful service for the application will receive messages. The configuration message to subscribe to a topic will not be broadcast over the network but will be forwarded only to nodes that are active and relevant to the topic.

MARINE [29] is a component-based middleware specifically designed for WSNs, which adopts REST and microkernel architectural patterns in its design. MARINE provides a communication service based on REST and, to deal with the dynamic environment and the need for resource optimization in WSNs, it provides inspection, adaptation and configuration services. New services can be specified by third parties and incorporated using the component model and programming interfaces provided by MARINE. The asynchronous communication service is provided by the PubSubHubbub protocol. MARINE creates a Hub on each sensor node so that every request to the node is taken directly to it, creating a high energy consumption since nodes are directly accessed. MARINE provides all the functionality required by a middleware for WSN. The main difference for PRISMA is that all requests pass through the gateway that is responsible for forwarding the request to a node that can provide the data and that has enough energy to complete the task avoiding the large energy consumption mentioned above. The gateway is responsible for determining which node should answer the request.

V. EVALUATION

In all experiments performed with PRISMA, the WSN comprised of Arduino Uno sensor platform that have 2KB RAM and 32KB of flash memory for program storage. This platform is powered by four AA (1.5V, 1500mAh) batteries that provide approximately 32 kJ of energy. The PRISMA was implemented using Arduino programming language. Experiments with real sensors were performed in the Ubiquitous Computing Laboratory of PPGI-UFRJ.

Considering the objectives of this work and following the methodology goal, question, metric (GQM) [30], we defined two goals. Goal 1 (G1): Analyze PRISMA with the purpose of evaluating its effectiveness with respect to meeting the programming abstraction feature for WSN

 ISBN: 978-1-61208-360-5

middleware in the context of application development and implementation. Goal 2 (G2): Analyze PRISMA with the purpose of evaluating its scalability in terms of the increase of application requests.

These goals were refined in four questions. Question Q1 is related to goal G1 and questions from Q2 to Q4 are related to goal G2. Q1: How expensive is it to build an application using PRISMA, in terms of lines of code? Q2: Does PRISMA scale well to serve a growing number of application requests? Q3: What is PRISMA overhead in terms of control/configuration messages? Q4: What is PRISMA overhead in terms of required RAM for its operation within WSN nodes?

The following metrics were defined to answer the questions considered in the evaluation. Each metric is denoted by M_{ij}, where *i* correspond to the question identifier, and *j* is a counter when there is more than one metric per question. The **number of lines of code (M_{11},)** is a metric used to evaluate how simple it is to create a sensing application using the abstractions provided by PRISMA (Q1). For computing this metric, we collected the number of lines of code required to create an application: (i) directly using Arduino programming and (ii) using PRISMA approach. The **Maximum number of requests supported (M_{21}):** is a metric used to assess whether PRISMA is scalable with respect to its programming abstraction approach, namely the use of REST (Q2). The **Time spent to deploy a new applications when PRISMA Web Server is overloaded (M_{22})**: is a metric used to assess the Gateway response time when a new configuration file is sent via the *Create* REST interface in a situation where many requests are made simultaneously. An increasing number of requests per second for the middleware interfaces were generated to determine the maximum number of requests supported and the delay expected to create new applications by varying the number of requirements sent to the middleware and thereby generating messages of varying size sent to the WSN. **The size of control message transmitted inside the WSN (M_{31})** is a metric is used to evaluate the overhead introduced by the control messages disseminated in the WSN (Q3). The RAM metric (**M_{41}) is used by sensing applications:** this metric is used to evaluate the overhead introduced by PRISMA (Q4). It verifies the RAM consumption when we use PRISMA to configure a sensing application.

A. Evaluation Methodology and Scenarios

To collect data to answer the questions an experimental evaluation was conducted. In the experiment, the network was planned so that it had two (2) clusters, each one containing a set of three (3) sensor nodes with different sensing capabilities, enabling the use of the topology control service at different times. A circular topology was organized where the sink node was at the center of the region, so that the sensor nodes and clusters remain equidistant from the center, thereby reducing the distance factor in latency and power consumption of both clusters. The radios of the sensor nodes were configured in order to respect the aforementioned topology. Nodes were hard-coded associated to their respective cluster heads and the transmission power was set as the same for all of them. Initially, all nodes had the same duty cycle. This cycle will only be changed by requests from client applications. Four client applications were developed, one producing periodic data requests and the other event-based data requests in order to collect the metrics specified. Each application represents a scenario. In the first scenario the application requests a periodic sample of temperature in the room "Lab1". The samples are to be collected every 15 seconds and the application will remain active for 5 minutes. The second scenario specifies an application that will execute for 5 minutes and collect periodic samples of the temperature, humidity and photo sensors. These samples will be collected every 15 seconds. The third scenario specifies an application that will execute for 10 minutes and collect samples of temperature every 15 seconds. However, in this case data will be sent only if the temperature exceeds 30ºC; moreover, the application requests a maximum delay of 200ms. The fourth scenario specifies an application that will execute for 10 minutes and collects temperature and photo samples every 15 seconds. Data will be sent only if the temperature: either exceeds 40ºC or is below 15ºC. The photo data will be sent by the nodes if the luminosity of the room exceeds 600 lumens. The maximum delay for this application is defined as 300ms.

B. Analysis of results

This section discusses the results obtained by extracting the metrics. The results are presented in Table 2. Regarding goal 1 (**G1**), the results of the metric **M_{11}** indicate, as expected, that the programming abstraction provided by PRISMA (creation of applications via an XML file and submission through REST interfaces) makes it simple to create new applications. In addition to reducing the number of lines needed to create an application, the client uses a higher-level language to specify the requirements. It is noteworthy that the difference in the number of lines increases with the complexity of the application created. In the table, A denotes Arduino and P mean PRISMA.

 ISBN: 978-1-61208-360-5

TABLE 2. EVALUATION OF RESULTS USING GQM

			Periodic				Event			
Goal	**Question**	**# Services / Metric**	**1**		**3**		**1**		**3**	
G1	**Q1**	$\mathbf{M_{11}}$	**A**	**P**	**A**	**P**	**A**	**P**	**A**	**P**
		(lines)	15	11	37	19	20	16	50	34
G2	**Q2**	$\mathbf{M_{21}}$ **(# requests)**	1200		950		1100		890	
		$\mathbf{M_{22}}$	111 ms / 114 σ		190 ms / 172 σ		186 ms / 153 σ		486 ms / 239 σ	
	Q3	$\mathbf{M_{31}}$	74 Bytes		171 Bytes		82 Bytes		195 Bytes	
	Q4	$\mathbf{M_{41}}$	Uno	Mega	Uno	Mega	Uno	Mega	Uno	Mega
		(bytes)	13332	14918	13332	14918	13332	14918	13332	14918

As for goal 2 (**G2**), the result of $\mathbf{M_{21}}$ metric indicates the maximum number of simultaneous requests before PRISMA Web Server component stops responding or demonstrates an unacceptable response time. We considered any response time above 800 milliseconds as unacceptable, following literature recommendations for typical Web applications. The result of $\mathbf{M_{22}}$ metric indicates the response time for deploying new applications through the *Create* REST interface provided by PRISMA. The response time was 246 milliseconds on average. Such response time in the literature is considered an imperceptible time from the point of view of typical Web applications. The response time for the creation of event-based applications is greater than the response time for creating periodic applications due to the higher number of transactions in the database. $\mathbf{M_{31}}$ and $\mathbf{M_{41}}$ metrics assess the overhead introduced by PRISMA. $\mathbf{M_{31}}$ measures the number of bytes transmitted in WSN nodes to create an application in each of the scenarios presented. We observed that the amount of bytes sent to the WSN to configure a new application increases according to its complexity. This increase is related to the number of messages that must be exchanged for the configuration of this new application. In the worst case, one message for each event/service requested is required. This happens due to the need of sending configuration messages to the cluster head that will select nodes that participate in the application and send this new configuration for these nodes. With respect to metric $\mathbf{M_{41}}$, the table shows the RAM consumption when using PRISMA on the Arduino UNO and Arduino MEGA. We can verify that the RAM consumption did not change between the scenarios and changed only between models. The variation between the models is due to the size of the bootloader of each model. It is worth noting that PRISMA consumed less than 50% of the available RAM on the Arduino UNO (32Kb) indicating that new services and features can be added to PRISMA.

Analyzing the results, we conclude that the complexity of the application affects the size of the XML necessary to create this application and the size of messages that are transmitted on the WSNs to configure this application. In contrast, PRISMA allows creating applications on the fly. This avoids redeploying the source code on the sensor nodes each time a new application arrives. The maximum number of supported requests and delay perceived by the customer are mainly affected by the characteristics of the hardware that was used to test and to implement the gateway.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented PRISMA, a resource oriented, publish/subscribe middleware for Wireless Sensor Networks. Results of a preliminary evaluation demonstrated the feasibility of implementing PRISMA in real sensor nodes and shown that it provides a suitable programming abstraction for WSN application development. This result points out that our approach is a "ready to use" middleware for an easy access, recent and open-hardware WSN platform. Moreover, the use of PRISMA architecture and REST interfaces allows future developers to continue evolving this approach by creating new services or clients to access data published by a WSN using PRISMA. For future works, we intent to evaluate the remaining features of PRISMA; in particular, its QoS mechanism that is basically provided by the topology control, as well as the asynchronous communication model introduced in this paper. We also plan to perform a comparative analysis with results obtained by Mires, and to analyze the impact of various parameters on PRISMA performance (e.g., number of sensor nodes, topology and application requirements). Finally, we intend to add support for actuators to cover a wider range of possible applications to use PRISMA.

ACKNOWLEDGMENT

This work was partially supported by Brazilian Funding Agencies FAPERJ, CNPq and CENPES.

REFERENCES

[1] TinyOS, “TinyOS.” [Online]. Available: http://www.tinyos.net/. [retrieved: May, 2014].

[2] Oracle, “Oracle.” [Online]. Available: http://www.oracle.com/br/index.html. [retrieved: May, 2014].

[3] Arduino, “Arduino.” [Online]. Available: http://arduino.cc/. [retrieved: May, 2014].

 ISBN: 978-1-61208-360-5

[4] M.-M. Wang, J.-N. Cao, J. Li, and S. K. Dasi, "Middleware for Wireless Sensor Networks: A Survey," J. Comput. Sci. Technol., vol. 23, no. 3, 2008, pp. 305–326.

[5] S. Hadim and N. Mohamed, "Middleware: Middleware Challenges and Approaches for Wireless Sensor Networks," IEEE Distrib. Syst. Online, vol. 7, no. 3, Mar. 2006.

[6] X. Koutsoukos, M. Kushwaha, I. Amundson, S. Neema, and J. Sztipanovits, "OASiS: A service-oriented architecture for ambient-aware sensor networks," Compos. Embed. Syst. Sci. Ind. Issues, vol. 4888, 2007, pp. 125–149.

[7] A. Taherkordi, Q. Le-Trung, R. Rouvoy, and F. Eliassen, "WiSeKit: A Distributed Middleware to Support Application-level Adaptation in Sensor Networks," in Proceedings of 9th IFIP Int. Conf on Distributed Applications and Interoperable Systems (DAIS), 2009, vol. 5523, pp. 44–58.

[8] P. Boonma and J. Suzuki, "BiSNET: A biologically-inspired middleware architecture for self-managing wireless sensor networks," Comput. Networks, vol. 51, no. 16, Nov. 2007, pp. 4599–4616.

[9] B. Valente and F. Martins, "A Middleware Framework for the Internet of Things," Conf. Adv. Futur. Internet, no. c, 2011, pp. 139–144.

[10] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," PhD Thesis University of California, Irvine, 2000.

[11] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," ACM Comput. Surv., vol. 35, no. 2, 2003, pp. 114–131.

[12] XML, "XML." [Online]. Available: http://www.w3.org/XML/. [retrieved: May, 2014].

[13] HTTP, "HTTP." [Online]. Available: http://www.w3.org/Protocols/. [retrieved: May, 2014].

[14] F. Delicato, F. Protti, L. Pirmez, and J. F. de Rezende, "An efficient heuristic for selecting active nodes in wireless sensor networks," Comput. Networks, vol. 50, no. 18, Dec. 2006, pp. 3701–3720.

[15] Apache TomEE, "Apache TomEE." [Online]. Available: http://tomee.apache.org/apache-tomee.html. [retrieved: May, 2014].

[16] MySQL, "MySQL." [Online]. Available: http://www.mysql.com/. [retrieved: May, 2014].

[17] Data Access Object, "Data Access Object." [Online]. Available: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html. [retrieved: May, 2014].

[18] ActiveMQ, "ActiveMQ." [Online]. Available: http://activemq.apache.org. [retrieved: May, 2014].

[19] Arduino Programming Language, "Arduino Programming Language." [Online]. Available: http://arduino.cc/en/Reference/HomePage. [retrieved: May, 2014].

[20] Wiring, "Wiring." [Online]. Available: http://wiring.org.co/. [retrieved: May, 2014].

[21] D. M. Ritchie, "The development of the C language," in The second ACM SIGPLAN Conf. on History of programming languages - HOPL-II, 1993, vol. 28, no. 3, pp. 201–208.

[22] Digi International, "XBee." [Online]. Available: http://www.digi.com/xbee/. [retrieved: May, 2014].

[23] A. Rapp, "XBee-Arduino." [Online]. Available: https://code.google.com/p/xbee-arduino/. [retrieved: May, 2014].

[24] M. Hart, "PString." [Online]. Available: http://arduiniana.org/libraries/pstring/. [retrieved: May, 2014].

[25] IEEE, "802.15.4." [Online]. Available: http://www.ieee802.org/15/pub/TG4.html. [retrieved: May, 2014].

[26] P. Boonma and J. Suzuki, "TinyDDS: An Interoperable and Configurable Publish/Subscribe Middleware for Wireless Sensor Networks," in Wireless Technologies: Concepts, Methodologies, Tools and Applications, A. M. Hinze and A. Buchmann, Eds. IGI Global, 2011, pp. 819–846.

[27] K. K. Khedo and R. K. Subramanian, "A Service-Oriented Component-Based Middleware Architecture for Wireless Sensor Networks," J. Comput. Sci., vol. 9, no. 3, 2009, pp. 174–182.

[28] E. Souto et al., "Mires: a publish/subscribe middleware for sensor networks," Pers. Ubiquitous Comput., vol. 10, no. 1, Oct. 2005, pp. 37–44.

[29] F. C. Delicato et al., "MARINE : MiddlewAre for Resource and mIssion oriented sensor NEtworks," Mob. Comput. Commun. Rev., vol. 17, no. 1, 2013, pp. 40–54.

[30] V. Basili, G. Caldiera, and H. Rombach, "The goal question metric approach," Encyclopedia of software Engineering, vol. 2, 1994, pp. 528–532.

 ISBN: 978-1-61208-360-5