A Survey on Adaptable Middleware for Wireless Sensor Networks

Edison Pignaton de Freitas
The use of sensor networks in different kinds of sophisticated applications is emerging due to several advances in sensor technologies and embedded systems and the increased demand from users. These systems run in highly dynamic and heterogeneous environments, in which changes occur very frequently, creating the need for adaptation support to environment changes and changes imposed by the users. An approach to allow the use of sensor networks in such complex and sophisticated applications is the use of middleware, especially adaptable middleware, which provides efficient response to the environment changes, adapting the middleware behaviour according to new requirements. In this survey we present a study of the state of the art in adaptable middleware for sensor networks, in which we analyse the main trends represented by important projects in the area, analysing their main features and providing a comparison among them.

**Keywords:** Middleware for Sensor Networks; Adaptable Middleware; Middleware Support for Applications in High Dynamic Environments.
SUMMARY

1 INTRODUCTION............................................................................................................. 7
2 CHARACTERISTICS AND CHALLENGES FOR WIRELESS SENSOR NETWORK MIDDLEWARE ................................................................................................................................. 9
  2.1 Wireless Sensor Network Characteristics .................................................................... 9
  2.2 Middleware Challenges .......................................................................................... 10
3 STATE OF THE ART MIDDLEWARE FOR WSN ...................................................... 12
  3.1 Middleware for WSN Approaches ........................................................................... 12
     3.1.1 Database-Inspired Solutions ........................................................................... 12
     3.1.2 Event-Based Solutions ................................................................................ 13
  3.2 Critical Analysis of the Traditional Approaches .................................................... 14
4 ADAPTABLE MIDDLEWARE FOR WSN ................................................................. 16
  4.1 DAVIM ................................................................................................................... 16
     4.1.1 DAVIM Architecture .................................................................................. 16
     4.1.2 Comparison with Other Approaches .............................................................. 17
     4.1.3 Prototype Evaluation ................................................................................... 18
  4.2 ATLAS .................................................................................................................... 18
     4.2.1 The Atlas Middleware ................................................................................ 18
     4.2.2 Analysis of the Applicability in the Embedded Network Systems ............... 20
  4.3 AGILLA .................................................................................................................. 20
     4.3.1 Comparison With Other Approaches ............................................................. 20
     4.3.2 The Agilla Proposed Approach .................................................................... 21
     4.3.3 Application Scenario ................................................................................... 23
     4.3.4 Performance Evaluation .............................................................................. 23
  4.4 IMPALA ................................................................................................................. 23
     4.4.1 The System Architecture ............................................................................ 24
     4.4.2 Application Programming Model .................................................................. 25
     4.4.3 Adaptation Support ...................................................................................... 26
     4.4.4 System Evaluation ....................................................................................... 27
  4.5 SINA ....................................................................................................................... 28
     4.5.1 SINA Architecture ..................................................................................... 28
     4.5.2 Information Gathering Methods .................................................................. 29
     4.5.3 Internetworking Between Mobile and Stationary Network ......................... 30
     4.5.4 Sample Examples ....................................................................................... 30
  4.6 TinyCubus .............................................................................................................. 31
     4.6.1 Application Requirements .......................................................................... 31
     4.6.2 TinyCubus Architecture ............................................................................. 32
  4.7 MiLAN ................................................................................................................... 34
     4.7.1 MiLAN Operation ....................................................................................... 36
     4.7.2 Medical Monitoring ..................................................................................... 37
  4.8 SensorWare ......................................................................................................... 39
     4.8.1 The Language ............................................................................................. 39
     4.8.2 The General Programming Model ................................................................ 40
     4.8.3 The Runtime Environment .......................................................................... 41
  4.9 TinyLime .............................................................................................................. 41
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TinyDB Graphical User Interface showing a query example</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>COUGAR Overview</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>High-level overview of DAVIM architecture</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>Atlas Overall Architecture</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>The Agilla Model</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>Mobile Agent Architecture</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>Agent Code Example</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>Higher-level Agent Code Example</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>The life cycle of a fire tracker agent</td>
<td>23</td>
</tr>
<tr>
<td>10</td>
<td>Layered system architecture</td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>Example of the even-based application programming model</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>Adaptation Finite State Machine</td>
<td>27</td>
</tr>
<tr>
<td>13</td>
<td>Clustering and Clustering Hierarchy Schema</td>
<td>28</td>
</tr>
<tr>
<td>14</td>
<td>Dispatching of incoming messages by SEE</td>
<td>29</td>
</tr>
<tr>
<td>15</td>
<td>Dispatching of incoming messages by SEE</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td>Dispatching of incoming messages by SEE</td>
<td>33</td>
</tr>
<tr>
<td>17</td>
<td>MiLAN employment</td>
<td>35</td>
</tr>
<tr>
<td>18</td>
<td>MiLAN Architecture Abstraction into the Network Stack</td>
<td>36</td>
</tr>
<tr>
<td>19</td>
<td>MiLAN Operation Overview</td>
<td>37</td>
</tr>
<tr>
<td>20</td>
<td>State-based Variable Requirements Graph Example: (a) Abstract; (b) Health Monitorig</td>
<td>38</td>
</tr>
<tr>
<td>21</td>
<td>QoS Graph Example: (a) Abstract; (b) Health Monitorig</td>
<td>38</td>
</tr>
<tr>
<td>22</td>
<td>SensorWare General Architecture</td>
<td>39</td>
</tr>
<tr>
<td>23</td>
<td>SensorWare Programming Model</td>
<td>40</td>
</tr>
<tr>
<td>24</td>
<td>Type of Events</td>
<td>41</td>
</tr>
<tr>
<td>25</td>
<td>Tuple Spaces of Mobile Hosts Connected</td>
<td>42</td>
</tr>
<tr>
<td>26</td>
<td>Operational Scenario Example</td>
<td>43</td>
</tr>
<tr>
<td>27</td>
<td>Main Architectural Componentes</td>
<td>44</td>
</tr>
<tr>
<td>28</td>
<td>Communication between Motes and BaseStation</td>
<td>45</td>
</tr>
<tr>
<td>29</td>
<td>Deployed Software Architecture</td>
<td>46</td>
</tr>
<tr>
<td>30</td>
<td>Operational Scenario Showing the Two Networks</td>
<td>46</td>
</tr>
<tr>
<td>31</td>
<td>Operational Scenario Presenting the Distinction Between Two Objects of the Same Type</td>
<td>47</td>
</tr>
<tr>
<td>32</td>
<td>Situation Previous and After Handover</td>
<td>48</td>
</tr>
<tr>
<td>33</td>
<td>Two Groups, Before and After Merge</td>
<td>48</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1 – CarTalk2000 and Sustainable Bridges Requirements Comparison ............... 32
Table 2 – A Big Picture of the Area of Adaptable Middleware for WSN .................. 51
1 INTRODUCTION

The use of wireless sensor networks in different kinds of sophisticated applications is emerging due to several advances in sensor technologies and embedded systems and the increased demand for context aware applications that rely on data provided by sensor networks. A specific kind of application that fits in this context is surveillance systems. In this kind of application, several different sensors are used. The sensors can be static on the ground or mobile, being embedded on mobile robots that move on the ground or in unmanned aerial vehicles that fly over the surveillance area.

Applications such as surveillance systems mentioned above have to face environments in which operation conditions change very often, requiring that the system be capable to adapt and reconfigure in order to respond to these dynamic scenarios. This response is not an easy task, as the nodes in the network are heterogeneous having different computing platform, sensor devices, and communication constraints.

A way to cope with the complexity in the development of applications that need adaptive behaviour is use a middleware platform that provides the necessary infrastructure and services that support the adaptations required to match new operation conditions. Many adaptable middleware exist, as described in (SADJADI 04), but when it comes to adaptable middleware for wireless sensor networks, the problem becomes more specific and the number of proposal are not as large as adaptable middleware for current applications (HENRICKSEN and ROBINSON 06).

Sensor networks have specific constrains that come from issues such as node heterogeneity, not only in terms of sensor capabilities but also in terms of mobility, as already mentioned, wireless communication, among others. Static nodes are generally resource constrained having all requirements related to energy consumption and rational resource allocation present in embedded systems. Mobile nodes have in general more powerful resources but they present other problems mainly related to communication but they can also present constraints related to energy consumption as it is desirable that they do not need frequent energy resource recharge.

Middleware for embedded systems could be employed in sensor networks; however they do not cover all the needs required by applications in this domain. The problems specific to the embedded character of the nodes in a sensor network would be solved, but those other problems related to communication and adaptation would not be covered. Besides that, there is also a lack of approaches that propose adaptable middleware for embedded systems. A recent work in the area that can be mentioned is (SILVA 08), but it does not address sensor networks yet.

Having all these singularities in mind, and observing that there is an open research area for adaptable middleware for sensor networks, this survey proposes a study about the main existing adaptable middleware for sensor networks looking for
their strengths and drawbacks, trying to draw a big picture of the area that can show open research questions that represent opportunities for further contributions.

The remaining parts of this work are divided as follows: in chapter 2 an overview of the main characteristics and challenges in the development of middleware for sensor networks is presented. This part of the work will allow comprehending the need for solutions that provide adaptation features in sensor network applications. Chapter 3 presents a brief summary of some state-of-the-art middleware for sensor networks that do not provide adaptive features. Chapter 4 cover the state-of-the-art adaptable middleware for sensor networks trying to analyse their pros and cons. In chapter 5 a big picture of the area is provided showing the main research questions that remain opened. Finally in chapter 6 present the conclusions of this study.
2 CHARACTERISTICS AND CHALLENGES FOR WIRELESS SENSOR NETWORK MIDDLEWARE

In order to understand the need for adaptability in wireless sensor networks, it is important to understand first the general needs involved in this kind of system. This chapter tries to summarize the main characteristics of wireless sensor networks and provide an overview of the main challenges in the development of middleware for this domain. The information provided in this chapter is mainly based in the (HENRICKSEN and ROBINSON 06), (MOLLA and AHAMED 06) and (YU; KRISHNAMACHARI and PRASANNA 05).

2.1 Wireless Sensor Network Characteristics

The operation of Wireless Sensor Networks (WSN) is in general quite different from operation of conventional computer networks. One of the main differences is the way to deploy the network. Nodes are spread over an interest area to monitor a certain phenomenon. These sensors can be static or dynamic, moving around the area or even flying over it. Sensing tasks are generally, defined at a high-level and the answers to them can be obtained by a combination of individual contributions of several limited sensors that compose the whole system.

Besides the particular way of operation, WSN also have certain characteristics that must be addressed in the design of a middleware that aims to support them. The first main characteristic is the limited resources available for some nodes that take part of the network. Nodes in WSN can be small devices that have limited amount of energy, restricted CPU performance, memory space, communication bandwidth and range.

The high degree of dynamicity is another important feature found in WSN. Failures, mobility and environment obstacles can disturb the functionality of the network.

Heterogeneity is also an issue that must be observed. Nodes can have different characteristics, like different processing power, amount of memory and available energy. This can be used to cope with the idea that all nodes in the network are small and resource constrained, but it also brings new needs in order to harmonize the interaction among different nodes.

Regarding those characteristics and operation modes of WSN, some software design principles are proposed, as the use of distributed algorithms to achieve a global goal of the system and adaptive fidelity algorithms used in order to achieve resource efficient use. Due to their use and the dynamicity of the nodes used in WSN, a new style of node addressing is also needed. This style focus on data produced by nodes, and not in the nodes themselves. This is called data-centric communication.
Data-centric or data-driven characteristic of WSN generates requirements for different routing protocols in which nodes are not addressed by identification address, but by the data that they provide. In fact, the node itself is not addressed, what is important is the data that it is capable to produce. As an example, in a sensor network that is used to monitor the temperature in a building, the queries are not addressed to nodes “X” and “Y”, but for a location, as “the conference room in the third floor” or “the place where the temperature is greater than a threshold”.

A very important characteristic of WSN is that they are in many cases deployed in harsh environments in which the possibility to modify some configuration or to change some software component must be done without physical access. It is the case, for instance, of the ZebraNet project (JUANG; OKI; WANG et al 02).

Still related to the environments in which the WSN are deployed, it occurs very often that these environments are very dynamic, in which several kinds of changes can happen. In order to namely some of them, change in weather conditions, movement of obstacles and crash of nodes are examples of events that can occur. The network must deal with this kind of problem in order not to loose its usability.

2.2 Middleware Challenges

The main challenges in the development of middleware for the WSN domain are presented in the following.

The first challenge in the adoption of a middleware approach for WSN systems is to define the basis over which it will be built. In conventional computers, it sits between the operating system and the application. However, sensor nodes do not have this clear separation in what is application and what is operating system. Many nodes execute tasks direct in hardware. This way, it is not easy to establish where the middleware will be placed.

Looking direct to the purpose of a middleware in WSN, it has to support the network development, maintenance, deployment and execution of sensing applications. It means that the middleware must coordinate the nodes distributing detailed low-level tasks, according to the high-level tasks that it receives, perform the merge of individual results issued by nodes and present the final report to the high-level user. The middleware must also handle with the heterogeneity of sensor nodes, providing appropriate abstractions and mechanisms to use different kinds of sensors regarding energy efficiency, robustness and scalability.

The middleware must also deal with the integration of the WSN and the conventional network with which it is linked. Classical mechanisms and infrastructures do not fit the needs to run in a WSN, but integration must exist between both networks. This way, the middleware designed to WSN must handle this concern.

Another important issue is that the middleware must provide mechanisms to inject application knowledge into the infrastructure, as it is a property that is presented by nodes in WSN.

The data-centric communication is a key concern in WSN and it fits better in an event-based communication paradigm than in a traditional request-reply scheme and a middleware that intents to be used by in WSN must offer this feature.

The high-level dynamicity and adaptability of WSN require new levels of support for automatic configuration (self-configuration), error handling and service degradation.
As several embedded system do, WSN also are used in application in which real-time requirements are present. Therefore, a support for handling of time constraint concerns must be offered by the middleware.

After this general presentation of the concerns related to the development of middleware for wireless sensor networks, a summary list describes briefly these ideas:

a) Abstraction Support: this is one the main goals of the middleware for WSN, in order to hide the underlying platforms (hardware and operating system), providing a homogeneous view of the entire network and simplifying upgrade or replacement of the platform;

b) Data Fusion: raw data collected by sensor nodes have to be merged or synthesized in order to provide a high-level and easily understandable format or report;

c) Resource Constraints: in order to work under very limited available resources, middleware for WSN must be lightweight;

d) Dynamic Topology: a key concern is related to the dynamicity of the topology in a sensor network, and a middleware for this kind of system have to handle this aspect;

e) Application Knowledge: in order to provide useful services to the final applications, the middleware have to integrate knowledge about the applications supported;

f) Programming Paradigm: programming paradigms for sensor networks, and also for the middleware services intended for these systems, are quite different from traditional ones;

g) Adaptability: adaptable behaviour is a key element in middleware for WSN in order to provide adequate support under operation environment changes, performance adaptation, among other;

h) Scalability: middleware for WSN must support scalability, not only in number of nodes, but also in number of user, as well as be able to operate over long periods;

i) Security: security issues in sensor networks, both in data processing and communication, are another key concern because the kind of applications in which they are used, like health care and military operations;

j) QoS Support: in order to ensure reliable services, middleware for WSN have to handle QoS requirements like resource and bandwidth allocation, response time, deadlines, among other.
3 STATE OF THE ART MIDDLEWARE FOR WSN

This chapter provides a summary of some state-of-the-art middleware for sensor networks that do not provide adaptive features, and by a critical analysis of these approaches, some open research directions are presented. This analysis tries to call attention for the need of adaptive behaviours in middleware for WSN.

3.1 Middleware for WSN Approaches

Most of the middleware developed for WSN are inspired by traditional middleware for conventional computer networks or in abstractions from other paradigms, like data-bases. This characteristic makes possible classify them according to the paradigm that they follow, and they are presented this way as follows.

3.1.1 Database-Inspired Solutions

In the first group, database-inspired approach, the selected middleware were the classical COUGAR (BONNET; GEHRKE and SESHADRI 01) and TinyDB (MADDEN et al 05).

COUGAR and TinyDB are designed for use in relatively simple data collection applications, only supporting very simple in network selection and aggregation functions based on simple arithmetic operations. Both have a SQL-like query language with support to temporal and data streaming. In Figure 1 the graphical user interface of TinyDB is presented in which an example of query can be observed.

![TinyDB Graphical User Interface showing a query example](image)

TinyDB is more sophisticated than COUGAR in the energy saving by calculating the frequency of the sampling to answer queries and also by the use of a routing
structure that helps the nodes to route in an energy-efficiently way. COUGAR uses a schema of leader nodes to aggregate data in the way back of data to respond queries. The overview of COUGAR structure is presented in Figure 2. In this figure it is presented the way a query goes to the sensor nodes via a query proxy (upper part of Figure 2), as well as the frontend and the user interface schematics (bottom part of Figure 2).

A key limitation in this two middleware is the assumption that sensor nodes are largely homogenous. The data types/relations that will be used at every node must be agreed in advance. This is acceptable in a small size sensor network; however it represents a great limitation for networks with more nodes and represents really a problem if a graceful evolution is required. A noteworthy limitation of the sensor database approach is that they are not prepared to support rich sensor types, as surveillance cameras with image processing. Another remarkable comment is about the use of a SQL-like languages not being appropriate as sensor data capture observations and not facts.

3.1.2 Event-Based Solutions

Event-based approach is based on the idea of publish/subscribe, allowing a decoupling of event producers and subscribers, however it supports additional
operations related to time and spatial conditions. The main drawback in the approach is the complexity involved in its implementation. Another problem is the assumption that all sensors report accurate data and that the set of sensors is homogeneous. DSWare (LI04) introduced some handling regarding uncertainty of events by the notion of confidence when looking at event correlations.

Mires (SOUTO et al 04) provides a more traditional publish/subscribe solution designed to run on TinyOS. In its architecture, sensors advertise the type of data that they can provide, while applications are able to select among these data, those in which they are interested. Sensors so publish the data to applications according to the subscriptions. However, no detail about how this protocol is implemented is provided.

3.2 Critical Analysis of the Traditional Approaches

By the analysis of the approaches presented the major problems and some key concepts that might be taken in account in the development of middleware for sensor networks can be summarized as follows.

a) Limiting Assumptions: all studied approaches make assumptions that limit their applications too much. Assumptions like the homogeneity of sensors, constrained resources, network topology are examples that limit the usability of those middleware to simple application and make them not suitable for complex applications envisioned;

b) Simple Features: based on those limiting assumptions, the middleware features offered by the analysed approaches are too simple, and do not meet the requirements of complex applications that need more than only a collection of raw environment data;

c) Context-aware and Sensor Network Convergence: Context-aware applications rely on data gathered by sensor networks. However, middleware in the area of context-aware domain are capable to support sophisticated queries and data fusion techniques that are not presented by sensor network middleware. The convergence of these two domains can enrich the support offered by sensor network middleware (HENRICKSEN and ROBINSON 06);

d) Query Abstraction: use of ontology can enrich the semantic of the data provide by sensor networks. One possible way to achieve it is using W3C standards like XML, RDF and OWL (HENRICKSEN and ROBINSON 06);

e) Information Quality: this is a concern present in the context-aware middleware that may be incorporated by middleware for sensor networks;

f) Adaptability: the emerging uses of sensor networks require the capacity of the applications face changes in the environment in order to continue responding the user needs. Another issue in relation to this topic is that the user requirements can change and the system must provide a way to follow this change and provide the desired data to the user;

g) Spread Intelligence over the Network: by enhancing features like data fusion and aggregation, it is possible to provide interpretation and reasoning in different locations of the network, what can provide better results not only in the direction of save scarce resources but sophisticating queries to match with the application requirements.

An explicit topic for adaptability was mentioned above. However it is a concern much broader as it is also related with context-awareness and distributed intelligence in
the network. In relation to the context-awareness, the real need for adaptation is considered because in a presence of a change of the operational conditions, if the system does not follow this change, the context-awareness will no more be achieved, as the it will be impossible to gather the data needed in a new scenario.

The distributed intelligence is another facet of the adaptability as the distributed reasoning over the raw data may need changes to follow a dynamic phenomenon that changes its position or its characteristics, for instance, requiring different handling each time a change occurs.
4 ADAPTABLE MIDDLEWARE FOR WSN

In this chapter we present the detailed analysis of some relevant work in the area of adaptable middleware for sensor networks. There are other works that were not considered in this survey, so it is not an exhaustive study; however we consider that the major trends in the area are covered by the work reported in this survey.

4.1 DAVIM

DAVIM (HORRÉ et al 08) is an adaptable middleware that enables dynamic management of services and isolation between simultaneous running applications over sensors networks. The authors make an analysis about the main issues a middleware designed to sensor networks might address in order to support end-to-end applications. They highlight the need for an easy composition of reusable functionality, separating services from applications, support for simultaneous applications running in isolated space, and dynamic management of available services.

The authors present a motivation case study in which a chemical industry has a sensor network used by three different applications. They present how each application may use the resources provided by the sensor network and how a decoupling between applications and services can benefit the overall scenario, as some services can be used by more than one application, without the awareness of each other. Besides that, they also highlight that if the basis of the system is adaptable, new applications can be built over it without any disturbance in the previous applications and services that were running before.

4.1.1 DAVIM Architecture

Based in the statements above, the DAVIM middleware approach is presented. Basically it is an architecture that uses virtual machines to run applications and services. Each virtual machine runs one application, and like this, the applications are isolated from each other. The services provided by DAVIM are implemented as the instruction set present for a specific virtual machine, according to the needs of the application that runs on it. Like this, instruction sets vary from one virtual machine to another. High level operations in a virtual machine instruction set are grouped in operation libraries. These libraries can be added, updated, or removed by the user at runtime. More formally, the architecture is divided in five building blocks, as shown in Figure 3.
Application store block manages application components. It operates with the coordinator to ensure that the application components stay up-to-date.

The VM controller is responsible for each virtual machine operation. It cooperates with the multiple virtual machine support.

Operation store supports the dynamic deployment of operation libraries. It keeps track of which operation libraries are load and cooperate with the installation, update, or removal of an operation library.

The VM store block contains all information related to the virtual machines, implementing most of the support for multiple virtual machines.

Coordinator provides the functionality to disseminate application components and virtual machines in the network.

4.1.2 Comparison with Other Approaches

The authors also present a comparison of their work with related research in the area. In summary, the major addressed compared features were: (1) the proposed approach provides the possibility of updates in runtime without the disturbance in the entire system, in opposition to the Maté (CULLER and LEVIS 02), that need to disseminate a complete TinyOS (HILL et al 00) image with the modified virtual machine in the network; (2) DAVIM uses the concept of virtual machine to separate reusable services from the application using them, in contrast with the proposed in Dynamic Virtual Machine approach (BALANI 06); (3) Tenet proposes a similar separation of services and application presented by DAVIM, however it does not present update at runtime; (4) a report about the energy efficiency provided by the use of a combination of virtual machine code and native code presented in (HORRÊ et al 08) is also mentioned to support DAVIM approach.
4.1.3 Prototype Evaluation

A description of a prototype evaluation is provided with some data addressing issues like overhead, dynamic deployment of libraries and the use of multiple virtual machines. Related to the first criteria, DAVIM presents similar overhead during dissemination and installation of new bytecode scripts if compared with other approaches. The dynamic deployment of libraries presented better results as Bombilla, for instance, because of the dissemination of a complete TinyOS image needed by this approach if compared with the DAVIM approach that just need to load small modules to deploy additional libraries. The use of multiple virtual machines does not present a very good result if compared with the other approaches, but the authors defend the thesis that it occurred due to the small size of the test scenario, and that in a realistic business scenario the proposed approach would overcome the use of a single virtual machine, for instance. Another aspect related to this issue that the authors highlight, is the absence of disturbance in the other applications when an update is need in a specific application.

4.2 ATLAS

Atlas (KING 06) is architecture for Sensor Network based Intelligent Environments. The main goal of this architecture is to provide services and support to applications like health care assistance in intelligent houses equipped for assist the elderly.

The main goals that this architecture aims to achieve are:
- eliminate the need for system integration (self-integration of joining entities);
- decouple application development from the physical world of sensors and actuators;
- provide a programmable pervasive space;
- to be as general as possible in order to let the pervasive space accept a variety of entities without any special technology limitation.

The architecture is divided in 3 main levels: the Atlas sensor and actuator platform, which is a bridge liking the physical devices to the digital system; the Atlas middleware, and the service authoring tool. The presence of the middleware in this architecture was the fact that made this work takes part of this survey. However, before talking more specifically about the middleware, it is important to highlight some of the assumptions made by the authors:
- centralized server capabilities;
- use of the TCP/IP protocol;
- abundance of energy, no energy consumption constraint.

4.2.1 The Atlas Middleware

The Atlas Middleware was thought to meet some key features listed by authors and that follows:
- modularity and extensibility;
- based on a service-oriented architecture;
- resource sharing by the offered services;
- easy to use, in order to provide an easy programming of the system that will use it;
the use of established standards.

The middleware is composed by several components, as shown in Figure 4, and presented as follows.

The Plug and Play Nodes represent the abstraction of the real-world devices, sensors and actuators, and the firmware running in each node is responsible for seeking out the centralized server framework to exchange configuration data. After that, the node is registered in the middleware as a service to make possible its automatic discovery and access.

The main part of the middleware is based on the OSGi, which is a framework that provides a runtime environment for dynamic and transient service modules called Bundles. It provides functionalities such as life cycle management as well as service registration and discovery. It is characterized by a service-oriented architecture that decouples components, allowing them to collaborate in a dynamic way. Each node is represented in the middleware by an OSGi service.

The Network Manager is responsible for keeping track of all nodes in the network. A handshake occurs between the manager and a new joining node. The manager registers the device services and activates the node.

The Configuration Manager encapsulates all the methods for recording and manipulating node settings. This manager receives the configuration of a new node from the Network Manager, parses the file and accesses the Bundle Repository to get the references for service bundles required by the connected devices.

The Bundle Repository, which is a repository that manages the bundles of services required by physical nodes, eliminating the need of those devices store bundles locally.

Context Manager and Safety Monitor create a context graph that represents all possible states of interest in the “intelligent” space. The states can be desirable, transitional or impermissible, and the goal is to take actions to always lead the space to desired states and avoid impermissible ones.
An array of Communication Modules is also present in the Atlas middleware in order to provide more effective communications in a heterogeneous environment.

4.2.2 Analysis of the Applicability in the Embedded Network Systems

The Atlas middleware is enclosed in the Atlas Architecture depending on some components that are outside of its scope. This dependence makes its use be related to the use of the whole architecture, which was not thought to be used in embedded applications.

One of the main constraints is the assumption that the sensor nodes in the network have unlimited source of energy, what makes its use not reasonable for application with hard energy efficient consumption requirements as in sensor networks that are not enclosed in facilities like houses or buildings.

Another drawback of the Atlas middleware is the lack of real-time requirements handling. It makes impossible the QoS control in communication among nodes, as well as the timing requirements in actuators activation and use.

However, some interesting features can be highlighted as interesting ideas that can the used as inspiration to works that suites the Distributed Real-time Embedded (DRE) environment. One of these ideas is the use of OSGi, which provides support to use standard technologies. Another interesting feature is the self-discovering allowing nodes joining the network in transparent way.

4.3 AGILLA

Agilla (FOK; ROMAN and LU 05) presents an approach that allow users inject special programs in the wireless sensor network by using an abstraction of mobile agents that can migrate across the nodes performing certain application-specific tasks. This migration of agents makes possible the adaptation of the network to run different applications according to the demanding requirements.

In the introduction of this paper, a motivation for adaptation is presented. The first supporting idea is that the long deployment time of wireless sensor networks may require flexibility in order to make changes according to the user requirements that may probably change along the time. The other key supporting idea is the fact that wireless sensor networks are generally deployed in highly dynamic environments, which applications have to be flexible enough in order to continue being useful in changing scenarios.

In the following, the supporting idea for the needy of a middleware that helps in the adaption of the network is provided. The authors use as example the MICA2 motes running TinyOS operating system, and explain that this hardware and operating system platform offer limited support for changes, being necessary the manually changes to reprogram the nodes to perform certain required adaptation in the network, what is impracticable in large sensor networks that are constantly suffering changes. Hence, the middleware technology promises be an important tool to provide flexibility for wireless sensor networks.

4.3.1 Comparison With Other Approaches

Some of the middleware approaches that provide adaptation features are presented and analysed as follows. XNP and Deluge present network reconfiguration by changing the mote’s instruction memory. However, both approaches present the drawback of take a long time and consume a lot of energy to transfer an entire image over the network.
and flashing the instruction memory. Maté and Impala (LIU and MARTONOSI 03) are two approaches that divide an event-based application into modules that are flooded across the network. The former uses a virtual machine approach and the last, native code. The drawback of these approaches is that there is no control where the new program will be installed, in fact, it will be installed in all the nodes of the network, and like this, there is no way to run several applications in the network at the same time. This drawback also occurs in Deluge (HUI and CULLER 04). SensorWare (BOULIS; HAN and SRIVASTAVA 03) is an approach the provide code mobility by the use of scripts, however this scripts do not carry state information. SensorWare and Impala have a common drawback of being targeted to a powerful platform (iPAQ 3670).

4.3.2 The Agilla Proposed Approach

Focusing on Agilla, it is an approach based on Maté, but unlike this one, which divides applications into capsules that are flooded throughout a network, it allows the deployment of applications by injecting code by mobile agents. These agents can move around the network nodes in an intelligent way, by simply moving, or cloning themselves to different location in response to changing conditions in the environment.

The application in Agilla consists of mobile agents that can perform application-specific tasks. Their coordination is made by the use of tuple spaces and the nodes are addressed by their location. Figure 5 presents the Agilla model. In each node, up to four agents can run concurrently.

![Figure 5 – The Agilla Model](image)

The tuple space and acquaintance lists present in the Agilla model are used to facilitate inter-agent coordination. The tuple space used is based on Linda (GELERNTER 85), and is shared by local agents and remotely accessible. This approach provides a decoupled style of communication where one agent can insert a tuple and another can read or remove it using pattern matching. An acquaintance list is present in every node containing the location of all one-hop neighbour nodes. In order to populate and maintain this list, a grid topology approach is used.

The model used by the agents is based in a stack architecture that allows most instructions be represented by a single byte. The agents are written a low-level assembly-like language similar to that used in Maté. The architecture model of the mobile agent is presented in Figure 6, while in Figure 7 is presented an example of
agent’s code. This low-level language is seemed by the authors as a weak point of their approach. Recognizing this, they are investigating the use of higher-level language, which Figure 8 provides an example.

![Figure 6 – Mobile Agent Architecture](image)

![Figure 7 – Agent Code Example](image)

```java
void main() {
    String f = "fir";
    Tuple t;
    t.addField(f);
    out(t); // insert fire tuple;
    while(true) {
        ledToggle(RED);
        sleep(100); // sleep 100ms
    }
}
```

![Figure 8 – Higher-level Agent Code Example](image)

It is important to highlight that Agilla does not support a global tuple space that spans across multiple nodes due to the constraints related to energy consumption and bandwidth. Other problem is that this approach does only consider nodes with limited resources, and like this, does not provide support for more complex computations.
possible in more powerful nodes. The mobile agents are susceptible to message loss what interferes in the ability of agents migrate and clone.

4.3.3 Application Scenario

A Fire Tracking Application is provided as an example of case study. The scenario consists of a network of sensors that provides a perimeter around a fire place, tracking the fire progression. The fire fighter can inject a guidance agent that will lead him along a safe path to the fire. In order to simulate this scenario, two kinds of fire agents were used: one static and another dynamic. The static fire agent just inserts a fire tuple in to the local tuple space and sit in a loop blinking the red LED, indication that that node is on fire. The dynamic fire agent models a fire that epidemically spreads throughout the network. It clones itself onto a random non-burning neighbour and repeats the blinking. The tracker agent is models as shown in Figura 9. It checks repeatedly whether any of its neighbours are on fire. If none are, it performs a weak move to a random neighbour and repeats the process. If a neighbour is on fire, it enters in a tracking mode which consists of lights up a green LED in the node and enters in a loop that first determines the locations of all neighbours that are on fire. Then for each non-burning neighbour that does not have a tracking agent within a certain region, it performs a weak clone.

![Figura 9 – The life cycle of a fire tracker agent](image)

4.3.4 Performance Evaluation

A performance evaluation of the approach applied in the described scenario shows that the tracking agent can spread over the network and form the perimeter in short time intervals in the majority of the cases, ranging from 3 up to 9 seconds, with more occurrences near to 3 seconds. A fact that could be noticed is that the initial point in which the tracking agent is injected has a significant impact on the speed at which the perimeter is formed. It occurs because there areas in scenario configuration in which the agents could not spread in parallel and just one node could access another, for instance.

4.4 IMPALA

Impala (LIU and MARTONOSI 03) is a middleware for sensor networks that enables application modularity, adaptability and reparability. This middleware allows software updates to be received via node’s wireless transceiver and to be applied to the running system dynamically. It also provides features to improve system performance,
energy-efficiency and reliability. It was developed as a part of a project called ZebraNet (JUANG, OKI, WANG et al 02) were sensor nodes are placed on free ranging wildlife to perform studies of animals in their ecosystem.

Some basic requirements of the network intended to be used in the ZebraNet project are: the long time of deployment; application may change during the use of the network, but the necessary change can be done with physical access to the nodes. These requirements can be translated in concerns like application adaptability, via updates of the running software. However, in order to do not affect the lifetime of the node, the updates can have strong impact on the energy consumption.

A monolithic application structure was analysed, and the drawbacks was pointed out. Even that this kind of structure generally provide a best performance, it is not flexible to provide the adaptability required by the ZebraNet. As an update was necessary, all the software would have to be changed, what could be done with physical access to the node, but it was not acceptable, or by the use of the network to transmit the update. However, the use of the network to transmit a full update would consume all the scarce bandwidth, and would require much of the nodes energy. Based on these statements against the monolithic structure, a modular one was chosen.

The modular structure offered by Impala allows the update of certain parts of the systems, without stop the running application. Another advantage is the reduced consume of the energy in transmitting updates, due to the smaller amount of data contained in the modules updates, if compared with an entire monolithic software.

The design rationale of Impala can be resumed as follows: (1) Modularity: the middleware is responsible for switching decisions, and like that, the applications can be independent and do not need to coordinate with each other; (2) Correctness: programming individual applications is simpler than a single big application; (3) Ease of Updates: Software changes can be done in small pieces of software, the modules; and (4) Energy Efficiency: Instead of transmitting an entire monolithic program, the updates can be transmitted at the granularity of smaller modules.

### 4.4.1 The System Architecture

Impala architecture is divided in two layers, as shown in Figure 10.

![Layered system architecture](image)

**Figure 10** – Layered system architecture.

The upper layer hosts all application protocols and programs. Only one application can run at a time, there is no support for concurrency in this middleware. The lower
layer contains three agents, the Application Adapter; the Application Updater; and the Event Filter. The first is responsible by the adaptations in the applications protocols face different operation conditions. It aims to improve performance, energy-efficiency, and robustness. The Application Updater is in charge of receives, propagates and install the updates. The Event Filter captures and dispatches events the above system units. Five types of events are supported: (1) Timer Events: it signals that a timer has gone; (2) Packet Event: it signals that a network packet has arrived; (3) Send Done Event: it signals that a packet has been sent or has failed to send; (4) Data Event: it signals that a sensed data from a sensing device is ready to read. (5) Device Event: it signals a detection of a device failure. The events are processed sequentially. They are bound in the time to be handled in order to avoid delay in handling the other events that might be waiting.

4.4.2 Application Programming Model

Impala uses an event-based programming model. Applications must implement four event handlers: timer handler; packet handler; send done handler; and data handler. The provided user library contains a number of general programming utilities, such as networking utilities that allow applications to send asynchronous messages. Timer utilities are also provided and allow the set up of timers for various purposes, like to send periodic messages, for instance.

Impala has global data structures that include uniform storage image of the sensed data and an execution frame to save the application execution states.

Figure 11 provides an example of the event-based application programming model. The timelines show a series of events and event handler actions between two communicating sensor nodes.

More details about each agent are provided in the following.
4.4.3 Adaptation Support

In order to provide adaptation, the Impala middleware counts with adapters that perform necessary adaptation, as they are demanded.

Adaptations are desired in two scenarios: First, as performance, energy-efficiency and other attributes of a protocol depend heavily on a number of factors, the adapter is responsible for the handling of several parameters that represent those factors and adapts to sensitive changes in their values. Second, failures in different devices can handle, but these devices are required by different applications, being critical for ones but not for others. The adapter improves the robustness by adapting the protocol choice of device.

The adaptations are implemented through event-based programming model, occurring in response to certain events. The adapter is responsible for the application switching decisions. The mechanisms and adapters that perform them are presented as follows.

4.4.3.1 Application Adapter

The goal of the Application Adapter is to achieve an effective software update mechanism for the mobile wireless sensor network. Some characteristics of this kind of network must be taken in account in order to achieve the proposed goal, such as high node mobility, constrained network bandwidth and wide range of updates.

The above mentioned characteristics implies in some issues that must be handled, such as: (1) incomplete, contemporaneous and inconsistent updates; (2) update vs execution; (3) update propagation protocol; and (4) code memory management.

The first issue is handled by the adoption of a scheme of software version numbering that allow the maintenance of the consistency, even face incomplete updates. The code memory stores complete and incomplete applications. An application is considered complete only when all its modules have been received.

The second problem related to the execution vs update, is handled by the installation of a new application with impact to the running application. This is possible by the use of an auxiliary record list for the new applications. As soon as they are installed, the record corresponding to the new application is copied to another table that stores the installed applications. If the application being updated is running, the update is completely done in the auxiliary space without interference in the running application. As soon as the update is complete, the running application is stopped, the new record is stored and the application is restarted with the new version.

In order to address the update propagation issue, the Impala adopt an on-demand software transmission strategy, based on the version number of the updates. The strategy has three phases. In the first all nodes advertises its software version information. After receiving the information of the other nodes, the node determines if its version is older or not. In affirmative case, it will send a request for the modules that it does not have. In the final phase, the node with the highest version sends the update to its neighbours.

The forth problem related to the memory management is solved by discarding incomplete updates, from the lower version to the highest in a same application, and from the same application to other applications.
4.4.3.2 Functionality Adaptation

The adapter has a table that keeps track of the state of the system by means of a set of application and system parameters. Application parameters include number of direct network neighbours, amount of sensor data successfully transferred and amount of free storage for application data. System parameters include battery level, transmitter power, range and data rate, and the geographic position of the node. Each application is responsible for updating a set of application parameters. By checking the current active application, its respective application parameter values, and the system parameters, the adapter examines the switching rules; if some of them is satisfied, an application switch is performed.

According to the parameter values, adapter makes adaptation decisions based on an Adaptation Finite State Machine (AFSM), which an example is shown in Figure 12.

4.4.3.3 Device-based Adaptation

Adaptations due to device failures are done in a very similar way as adaptations in functionality. The middleware has a table that relates the applications to the devices required by them. In an occurrence of a failure, the adapter responds by disabling the protocols that require the failed device.

4.4.4 System Evaluation

Impala was evaluated running over an iPAQ handheld platform as a testbed and by a simulation of a complete network in the network simulator ZnetSim.

The use of the on-demand approach for the update propagation showed an efficient network reprogramming if compared with a broadcast propagation, infecting the network in a comparable time but with software transmission volume much lower than the broadcast transmission in the testbed.

Other benefits were mentioned in the report of the evaluation, based on the simulation done. The first is the improvement of the routing performance, by adapting the range of the transceiver based on the network connectivity. The rate of success in the transmissions using the adaptations (switching between long and short range transmissions) shows better results than using only short range or only long range transmissions. The energy efficiency is also improved by the use of the adapting protocol even that the total amount of energy consumed by this protocol is larger than the short range, which consumes less. It can be said as the use of a protocol that consumes less (short range) with no success in the transmission can be seen as a waste of energy. On the other hand, using adaptation, the rate of success increase because in the cases in which the use of short range would be useless, it was switched to the long range, achieving success in the transmission.
4.5 SINA

SINA (SHEN; SRISATHAPORNPHAT; JAIKAEO 01) is a database oriented middleware to sensor networks that facilitates querying, monitoring and tasking by abstracting the network of sensors as a collection of massively distributed objects. It has an execution environment that allows scalable and energy-efficient organization and interaction of sensors. The data access is done by the use of a SQL-like language called SQTL (Sensor Query and Tasking Language) and the tasks by programming scripts.

4.5.1 SINA Architecture

SINA modules running in each node provide adaptive organization of sensor information facilitating query, event monitoring and task capability. Sensor nodes are autonomously grouped in clusters, what provide the scalability and energy-efficient support. SINA presents three functional components, as described as follows:

Hierarchical Clustering: Nodes are grouped in clusters according to their power levels and proximity. These clusters form a hierarchy recursively providing levels or tiers of clusters. Within a cluster, a cluster head is elected. This node is responsible for filtering, data fusion and aggregation. If a cluster head fails, a new election occurs. Figure 13 shows the clustering schema.

Attribute-based Naming: As an identification of nodes by identifiers is not scalable and difficulty to interface with the final user, an attribute-based naming scheme is used. It explores the data-centric characteristics of sensor queries.

Location Awareness: It is important that nodes have knowledge about their own physical location as the networks can be deployed in large areas. The use of GPS is considered, but not only the simple use of a GPS in each node, due to the increase in costs, but also the use of beacons that have GPS receivers and are used by other nodes that do not have GPS, but infer their own position by the position of the beacon nodes and their relative distance. Other considered alternative is the use of optical trackers, but it has the drawback of the effective only in a small region.

The information abstraction present in SINA is based on the idea of view the network as a collection of datasheets; each of these datasheet contains a collection of attributes of each node. Cells are created in the nodes according to requests for information from other nodes. Each created cell is uniquely named and becomes a node’s attribute.

Figure 13 – Clustering and Clustering Hierarchy Schema.
An important part of the architecture is the language used by SINA. The SQTL is the programming interface provided by SINA to the applications. It is characterized by a procedural scripting language, flexible and compact. SQTL supports hardware access, location-aware, and communication primitives. Moreover, it also provides event handling constructions supporting three kinds of events: events generated when a message is received by a sensor node; events triggered periodically by a timer; and events caused by the expiration of a timer.

In order to target a script to a specific receiver or group of receivers, the message have to be encapsulated in wrapper which represent a header of the SQTL message, indicating the sender, the receiver, the application running in the receiver, and the parameters for the application. The syntax adopted by SQTL wrapper is XML.

In each node a sensor executing environment (SEE) runs, being responsible for dispatching incoming messages, examining all arrival SQTL messages and performing the appropriate operation for each type of action specified in the messages. Messages have some key arguments like “ALL_NODES” that describes a broadcast message, and “NEIGHBORS” that describe a message that may be forwarded only for one-hop distance nodes. Other fields are used in the control of the messages, like the criteria, that is used to verify if the information type requested matches with the data available in the local sensor datasheet. A message with argument tell, is generated after a result is produced and is forwarded in the way back from which the request came. Figure 14 shows the dispatching of incoming messages by SEE.

![Diagram](image)

Figure 14 – Dispatching of incoming messages by SEE.

### 4.5.2 Information Gathering Methods

Three information gathering methods are provided in the SINA approach to overcome the response implosion problem that may occur when a number of sensors respond to a certain request (IMIELINSKI and GOEL 00). These methods are described as follows:

Sampling Operation: This method consists of a decision making by the nodes if they should participate or not in the response, based on a given response probability. The idea is that not all nodes need to respond if a number of its neighbours do. The probability is computed by the cluster-head according to the nodes present in the area and number of replies required from each cluster. This operation is called adaptive probability response (APR).
Self-Orchestrated Operation: This is a method used in networks with small number of nodes and that need the response from all nodes to provide accurate results. In order to avoid the implosion problem, different delays in response sending are used by each node.

Diffused Computation Operation: An aggregation scheme can be programmed by the SQT data dissemination among sensor nodes in order to route information back to front-end node, avoiding information implosion that would occur if all nodes sent information direct to the front-end. A negotiation-based data dissemination mechanism called SPIN (HEINZELMAN; KULIK and BALAKRISHNAN 99), can also be applied by SINA.

4.5.3 Internetworking Between Mobile and Stationary Network

A problem to be solved in the use of mobile nodes requesting information to stationary networks is how to route the response back to the requester node. Two mechanisms are possible: the first assumes that the information about the location of the mobile node has to be updated in the node that received the request (called query resolver). This approach has the problem that the location information has to traverse all the way to arrive in the resolver. The second is that in which the mobile node leaves footprints along the way by sending advertisements informing the nearby sensor nodes. It is called progressive footprint chaining and does not have the drawback of the first alternative. Figure 15 presents both approaches.

![Figure 15 – Dispatching of incoming messages by SEE.](image)

4.5.4 Sample Examples

Two sample applications are presented to evaluate the SINA middleware. One with just stationary nodes with the aim to perform network diagnosis; and another representing a coordinated vehicle tracking application in which a vehicle crosses the network.

In the first application, the three gathering methods were used to perform the system diagnosis, the Centralized sampling and self-orchestration; the APR and the Diffused Computation. The presented result showed that the use of the diffuse computation achieved the better results. These results were enhanced with a composition of sampling in the method. It is important to highlight that the distribution of computation was done by SQT data dissemination.
principle is to have most of the sensors in a stand-by mode, and as the target moves in the direction of a node, the node that is tracking the target send a retracking message to wakeup the node that is in the way of the target. By doing that, nodes that are not in the way are not waked up and the nodes by where the target already passed can enter in stand-by node. The results of this sample application showed significant saving in the energy due to the useful sensing rate compared to the total of sensing, however, it showed also an increase in the messages exchanged by nodes, and it is explained by the control messages to wake up nodes. Another interesting fact is that it is important to observe the speed of the target and analyse if the time elapsed by the sending of a wake up message and the activation of a node allows the continuous tracking or if it is possible to loose the track due to this delay.

4.6 TinyCubus

TinyCubus (MARRON et al 05) is not a middleware, but a framework that aims to provide the necessary infrastructure to cope with the complexity in developing applications for sensor networks. We are considering this approach in this survey, even that the related work is not exactly a middleware, because the considered requirements and the goals related match with those that are presented in adaptable middleware for sensor networks.

The proposal of TinyCubus is a flexible and adaptable cross-layer framework that provides a distribution of components efficiently by considering the roles of sensor nodes, supporting the dynamically installation of components and data management that selects and adapts both system and data management components.

In the proposed approach heterogeneity in the network is handled by considering the presence of resource-constrained nodes and mobility even in high speeds. Differences in functionalities are also taken in account as well as change in the requirements what bases the need for adaptability as related in (MARRON et al 05).

The general architecture of TinyCubus is composed by:

a) a cross-layer framework that supports data sharing and interactions between components in order to achieve cross-layer optimizations;

b) a configuration engine that allows reliable code distribution taking in account network topology and nodes functionality;

c) a data manager framework, which supports the dynamic selection and adaptation of the system and data management components.

In order to illustrate to usefulness of the proposed framework, two different applications are compared and how the proposal deals with those differences is analysed.

In the following it is presented the summary of the requirements of the mentioned applications and after that, the details about the architecture are provided.

4.6.1 Application Requirements

As canonical examples for the TinyCubus development two applications are considered, one is the Sustainable Bridges (SUSTAINABLE BRIDGES 08) and the other is Cartalk 2000 (REICHARDT 02).

The Sustainable Bridges project aims to use a sensor network to monitor bridges with static nodes in order to detect structural defects. Different kinds of sensors are used, such as temperature, humidity, vibration as well as noise detectors. In order to
determine the location of defects, noise is sampled and by using triangulation, the specific location is determined.

In the Cartalk 2000 project a cooperative driver assistance system is proposed in order to offer a warnings related to traffic jams and accidents. Queries about the road conditions and vehicle speed can also be requested. The sensors are integrated in the cars, so the can move in relation to each other and so an ad-hoc network take place.

Comparing these two applications, several differences are highlighted, like the mobility issue that is a worry in the Cartalk 2000, but it is not for the static nodes of the Sustainable Bridges project. Other issues like the network topology and energy consumption are very important when monitoring a bridge, but it is not that important in an ad-hoc vehicle network. The way the information is queried and how nodes announce their services also differ, as in a vehicle ad-hoc network a query-reply mechanism is more used if compared with a network that monitors a bridge and only when an event occur it issues information to the user interface, and it is not useful query the network all the time in this case. Table 1 presents the summary of the comparison between these two applications.

<table>
<thead>
<tr>
<th>Property</th>
<th>Sustainable Bridges</th>
<th>Cartalk 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Model</td>
<td>Specific</td>
<td>Generic/flexible</td>
</tr>
<tr>
<td>Query Model</td>
<td>Push-based</td>
<td>Pull-based</td>
</tr>
<tr>
<td>Progr. Paradigm</td>
<td>Publish/Subscribe</td>
<td>Generic query-based</td>
</tr>
<tr>
<td>Distr. Transp.</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Energy</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Mobility</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Time Sync</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Topology</td>
<td>●</td>
<td>○</td>
</tr>
</tbody>
</table>

○ Not important ● Medium ● Very important

The differences were well described; however, in order to support a claim for a unique framework that fits different application like that, it is important to highlight the common features that are needed in both cases. These are applications are either data-centric or data-driven. They are also state-based, as they might change depending on the current state of the application. Fault-tolerance is also an important aspect in both of them, as nodes can failure and environmental conditions can change. The guarantee of the availability of information most also be provided, as well as the possibility of adaptation and reconfiguration of software running in the nodes. Based on these appointments, it becomes clear that a generic framework is really a good deal to support different applications. However, in order to achieve this goal the framework has to be extensible and flexible to fit in different domain. Moreover, it must provide a parameterization of generic components to fulfil specific requirements.

4.6.2 TinyCubus Architecture

As mentioned before, the architecture of the TinyCubus consists of three parts, the Cross-layer Framework, the Configuration Engine and the Data Management Framework. Figure 16 presents this general architecture and in the following, more details about each of these parts are provided.
4.6.2.1 The Cross-layer Framework

The goal of this part is to provide a generic interface to support parameterization of components using cross-layer interactions. Like this, interactions among different layers are possible, and not only with immediately neighbour layers. This framework also provides custom code execution, by using the ability to execute dynamically loaded code.

An important part of the cross-layer framework is the State Repository, which in fact allows that the interactions among layers do not affect desirable architectural properties, like modularity. In this way, the framework acts as a mediator between the interacting layers, and the data are not directly accessed from one layer to another, but it is stored in the state repository. If a component is replaced, the interacting components are not affected, as the new component will use the same or compatible data in the state repository.

Some challenges faced by at this level are mainly the possible loose of modularity by the layers interaction and way that the parameters are stored in the state repository.

4.6.2.2 The Configuration Engine
It consists of a topology manager and a code distribution facility. The self-configuration of the network is possible due to the assignment of roles to each node that depends on its functionality. The code distribution installs components dynamically on the nodes, using role information to efficiently do that.

The topology manager assigns a role based on the nodes properties, such as hardware capabilities, network neighbourhood, location, etc. Examples of roles are: SOURCE, AGGREGATOR, CLUSTERHEAD, VIBRATION, etc. In order to do this, it uses a specification language and decentralized algorithm that are described in details in (RÖMER 04).

Code distribution is based on transmission of individual code fragments and the atomicity of code updates within a node.

This part also faces some challenges like the harmonization between the role assignment language and the implementation language used by TinyCubus, nesC. Another problem is related to the dynamic installation of components on the nodes.

4.6.2.3 The Data Management Framework

The goal of this part of the proposal is to provide a set of standard data management and system components and to choose the best set of components based on three dimensions: the system parameters; the application requirements; and the optimization parameters. The cube in the middle of Figure 16 illustrates this idea.

System parameters encapsulate characteristics like density and mobility of nodes. Application requirements are related to requirements such as reliability. Optimization parameters are related to some constrains or optimization criteria as minimal energy consumption. A certain algorithm is chosen based on the commitment with these three dimensions requirements. The one that suites best is selected by the Data Management Framework.

One highlighted problem is how to select a component if more than one fits the needs, and further, how to select if the applications requirements and system parameters of two different components fit but they support different desired optimization parameters, such energy one and latency the other.

4.7 MiLAN

MiLAN (Middleware Linking Applications and Network) (HEINZELMAN et al, 04) is an adaptable middleware that explores the concept of proactively adaptation in order to respond the needs in terms of QoS in sensor networks imposed by changes in the operational environment. The authors of this middleware says that a major drawback in the most of existing middleware for sensor networks is really the fact that they just provide reactive adaptation, and in dynamic environments as those in which WSN are deployed, this behaviour does not cover the real needs for adaptation required by running applications.

This middleware allows the applications specify their required quality for data, adjusting the network to increase its lifetime in the same time that the quality needs is still being meeting. The information that MiLAN receives from applications are:

- Individual QoS requirements over time and how to meet these needs by using different combinations of sensors;
- Relative importance about each application;
- Network status, such as available sensors and their resources (i.e. remaining energy and available bandwidth).
These received information is used by MiLAN to continuously adapt the network configuration, selecting which sensors should provide the required data, which sensors should be routers in multi-hop communication and which of them should play special roles in the network, such as sink or information aggregator. Figure 16 shows a system with employment of MiLAN. The squares with a letter “M” inside represent the MiLAN that is implemented in each sensor. The different shapes attached to them represent different kinds of sensors. In the right side of the picture it is possible to see the individual applications informing their specific QoS needs and the middleware replying with the desired data, as well as the overall system providing general QoS information, such as the relative importance of each application to the middleware.

![Figure 16 - MiLAN employment](image)

MiLAN do not come with new implementations of network and MAC layers, instead, it is built on top of the existing protocols. The energy savings and the QoS requirements meeting is achieved by varying the parameters of the network by using the available resources provided by the underlying protocols. Unlike traditional middleware that sits between the application and the operating system, MiLAN has a different architecture that extends into the network protocol stack. Figure 18 presents this architecture.
4.7.1 MiLAN Operation

MiLAN receives a description of application requirements, monitors network conditions and optimizes sensor and network configurations to meet the QoS requirements while maximizing the application lifetime. Figure 19 shows an overview of the interaction among MiLAN and applications and sensors. In this picture, it is possible to observe the distinction between the network plugins and the core of MiLAN, where there is the separation of computation that is specific to the selected network type and the computation that always occurs. $F_N$ and $F_A$ in the picture correspond respectively to the subset of nodes that can be supported by the network and the subset of nodes that can provide data with QoS equal or greater than the required by the application. The intersection of this two set gives the nodes that can be used by the network to respond to the application needs.
In the following, an application consisting of a medical monitoring sensor network will be explained in order to give more details about how the middleware works.

### 4.7.2 Medical Monitoring

Consider a personal health monitoring application running on a PDA that receives and analyses data from a number of sensors (e.g., ECG, EMG, blood pressure, blood flow, pulse oxymeter). The monitor reacts to potential health risks and records health information in a local database. The QoS requirements related to information provided by each of these sensors varies over time with changes in the patient’s state. For example, higher quality might be required for certain health-related variables during high stress situations such as medical emergency and lower quality during low stress situations, such as sleep.

The performance of this application can be assessed by the QoS of different variables of interest, where it is clearly dependent of the sensors that provide the data related to the variables. Some variables such as blood pressure and respiratory rate can be assessed directly by the data of one sensor. The blood pressure sensor can though provide a quality 1.0 (100%) in determining the variable blood pressure. However, it can give less than 1.0 in determining other variables such as heart rate, which can be improved by a high-level fusion with data from the blood flow sensor.
In order to determine how best to serve the application, MiLAN uses the following information: (1) the variables of interest to the application; (2) the required QoS for these variables; and (3) the level of QoS that each sensor in the network can provide for each variable of interest. In the beginning, MiLAN receives the information required by the application in terms of a “State-based Variable Requirements” and “Sensor QoS” graphs. In Figure 20 is presented an example of a State-based Variable Requirements graph, in which the part “a” shows an abstract example while the part “b” presents an example related to the health monitoring application. In Figure 21 a Sensor QoS graph is presented, again with an abstract example in part “a” and part “b” with an example from the health monitoring application.

Given the information from these graphs as well as the application state, MiLAN can determine which sets of sensors can be used to satisfy the QoS needed by the application for each variable of interest (set $F_A$). However, MiLAN must also gather information from the network related to its specific properties and current condition to determine which sensor from the network can actually be used to respond to the application needs. In order to perform this task, MiLAN uses a service discovery protocol (such as SDP (AVANCHA; JOSHI and FININ 02) or SLP (IETF 08)) to find nodes and learn about their state. Basic information gathered by MiLAN are: (1) type of data that the node can provide; (2) modes of operation; (3) transmission power; and (4) residual energy. Based in this information the set $F_N$ is determined and by the intersection with $F_A$ the sensors to be employed to serve the application are chosen. If the intersection is empty, the middleware returns an exception to the application that can though decide best action to take. If more than one sensor can perform the same task in
the resulting intersection, a best performance/cost trade-off must be considered. It depends on the application that can decide the optimizing criteria to be used in the decision.

4.8 SensorWare

SensorWare (BOULIS 07) defines and supports a lightweight and mobile control scripts that allow an efficient use of the computation, communication and sensing resources at the sensor nodes. It is done by the use of service abstraction that can change in runtime by dynamically defining new services.

The approach adopted by SensorWare is based on active sensors allowing any distributed algorithm to be executed in the network. The programs, represented by scripts, see the resources of the nodes as services that can be shared among different programs running in the node. The mentioned dynamic installation of new services is seemed as one key feature of the proposed approach, due to the adaptation capability that it confers to the network.

The SensorWare proposal consists of two parts, the language and the supporting runtime environment. The general architecture is presented in Figure 22. The SensorWare layer uses functions and services offered by the OS to provide the runtime environment needed by the scripts that run over it. Scripts in a node can communicate with scripts allocated in different nodes. In the following some details about the language and the runtime environment are provided.

![Figure 22 – SensorWare General Architecture.](image)

4.8.1 The Language

The basic idea is that mobile scripts control the nodes. In order to do that, two basic needs are identified: 1) the language need commands to act as basic building blocks for the scripts; 2) the language must be constructed in order to tie these building blocks together in order to form actual scripts, allowing the construction of control flows, such as loops and conditional statements, construction of variable handling, and expression evaluation. These constructs are called the “glue core” of the language. The choice was to use TCL (OUSTERHOUT 94) that suites well to the needs, offering a
great modularity and portability. Thus, the TCL is used as the glue core of the SensorWare language.

Basic commands and more specific ones are provided by the language, but as the authors identified the necessity to provide a mechanism to allow the language evolve, they provide the support to the creation of virtual devices. This way, all service abstractions are represented as virtual devices. There is a set of minimal commands to communicate with a device, they are: query, act, interest and dispose. The first asks the device for information, the second instructs the device to perform an action, the third describes and specific event that the device can produce and gives a name to this event, and the dispose, just dispose an event name. The expandability resides in the fact that the task definition and the parsing of arguments of the four commands are defined by the developer. It allows the expandability keeping the structured form.

4.8.2 The General Programming Model

Scripts in SensorWare are described as state machines influenced by external events. Based on that, the programming model is the following: an event is described and it is tied with the definition of an event handler. According to its current, this handler will perform some processing and possibly create some new events and/or modify a current state. This is graphically represented in Figure 23.

![SensorWare Programming Model](image)

**Figure 23** – SensorWare Programming Model.

The WAIT command is used in order to allow the behaviour described in the state machine presented in the Figure 23. The events that a script can wait are presented in Figure 24. The use of high-level events result in a more efficient code.
It is important to notice that the language proposed by SensorWare is not a node-level programming language. It is an event-based language in which the behaviours are not tied to specific nodes, but rather to possible events that depend on the physical phenomena and the network state.

### 4.8.3 The Runtime Environment

As explained for the language, the runtime environment of SensorWare is also expandable by means of the concept of devices. This way, new services can be dynamically added to the middleware.

A well-defined and structured way to provide services is offered in SensorWare. It is done by registering a device name, followed by the definition of the four basic device-interaction commands, and defining any device tasks. Scripts are used to perform these actions. The main part of the script implements the device tasks.

There are some non-device tasks, such as Script Manager. This script accepts all requests for spawning of new scripts and keeps any script-related state. Passing data (state) along with the script code is achieved through the commands CARRY and PARAMETER. The first passes a list of values to the script and the second assigns variable names to these values.

Some additional features are also available in the SensorWare, such as code safety, resource management, and script compression, as presented with more details in (BOULIS; HAN and SRIVASTAVA 03).

### 4.9 TinyLime

TinyLIME is a data-sharing middleware based on LIME (CURINO 05), which in its turn adapts and extends the mobile tuple space model presented in Linda (GELERNTER 85), for sensor networks. The main contribution of this work is the operation setting supported by this middleware which provides a self-managed data-sharing that by itself do not consist in an adaptation, but support response to network changes and node mobility what can be seem as an adaptable feature to provide data access.

The goal of this middleware is support mobile ad-hoc networks providing an illusion of a shared memory between applications and sensors. There is no central data collecting point, instead, mobile nodes move around the area in which the nodes are deployed and by a one-hop communication receive data of the nodes directly reachable.
As TinyLIME is based on Lime, many characteristics are inherited from that middleware. In the following the tuple space conception from Linda and some of the key features of LIME are presented, and after that, the TinyLIME operation setting, model and architecture are detailed.

4.9.1 Tuples and Tuple Space

The concepts of tuples is used to share data among nodes. Tuples are defined as sequences of typed fields that represent certain information to be shared by different process. Process coordination is done by writing and reading tuples. Tuples contain actuals or formals. Actuals are values, like in ⟨“foo”, 8, 2.5⟩. Formals represent types of data, such as integer or float, as presented in by the two last fields in ⟨“foo”, ?integer, ?float⟩. Formals are used as wildcards in order to select a tuple from the tuple space when they match with actuals.

The operations that provide coordination are basically out(t) and in(p). The first is used to put a tuple in the tuple space while the second is used to take a tuple that satisfies a pattern “p” from the tuple space. There is a non-destructive read operation, rd(p), which allow read a tuple without take it from there. Both rd and in are blocking. There are variants asynchronous, called inp and rdp.

4.9.2 LIME

In order to support nodes mobility, LIME breaks the Linda tuple space into multiple tuple spaces, each permanently attached to a mobile node, and define rules for the sharing of their content. The static global tuple space from Linda is, like this, reshaped into a dynamic tuple space that changes according the nodes connectivity. In Lime, the tuple space is referred as an interface Tuple Space. When nodes are connected, a dynamically changing federated tuple space is composed by nodes’ ITS, based on the connectivity. Figure 25 presents an example in which this federation can be observed. Moreover, in this picture it is shown mobile software agents and physical mobile hosts that make part of the Lime model. Agents are permanently assigned an ITS which is brought along during an agent migration and reside in the mobile hosts. Co-located agents are considered connected.

![Figure 25 – Tuple Spaces of Mobile Hosts Connected.](image)

LIME adds two new notions to Linda: tuple location and reactions. The first is related to the fact that even that the tuple is accessible by all connected agents; it does exist only in one place of the system. The other allows an agent to register an especial
kind of code, called listener, which is executed whenever a tuple matching a particular pattern anywhere in the federation tuple space.

4.9.3 TinyLIME Operation Setting and Model

The operation scenario presented is different from the one discussed in traditional centralized models. The alternative proposed does not require multi-hop communication among sensors, instead, it relies on a one-hop communication between a mobile base station and sparsely distributed ground sensor nodes that do not need to communicate with each other. In this environment, it is also considered the existence of hosts that are clients and do not have direct access to the sensors, but communicate with the base stations that provide the required data by transmitting the queries to the involved sensors.

In order to support this scenario, TinyLIME uses the transient shared tuple space to store tuples containing the sensed data. The sensor nodes (motes) communicate with a base station when it gets into their communication range. Motes are only seemed by TinyLIME when connected to bases stations. A mote has this way an agent residing in the base station host, with its ITS containing the set of data provided by its sensors.

A notion of scope is introduced. Tuples with data from motes linked with different bases stations can be seem by a client host. If a base station issues an operation, such as read, within the scope of itself, it will get data only from the motes that are connect to it. Instead, if the host client issues the same read operation but with unrestricted scope, it receives data from motes linked to the two base stations. A concrete example of this is illustrated in the Figure 26, where the PDA represents a client host, and the laptops base stations linked to some motes that are in its range.

![Figure 26 – Operational Scenario Example.](image)

Reactions works as in LIME, with the extra limitations of connectivity already explained, but with an additional expressive power. It can specify the rate in which the data is to be refreshed and a condition to trigger the reaction, such as a range of values of interest.
4.9.4 TinyLIME Middleware Architecture

TinyLIME model was designed and implemented for the Crossbow Mote platform, exploring the TinyOS functionalities. Figure 27 presents the main components while a detailed description of them as well as of their interaction is provided in the following.

**Client Component**

A client interacts with the sensors through the MoteLimeTupleSpace class, which extends LimeTupleSpace from the LIME API. The format of tuples containing sensor data is predefined and contains four fields: `(SensorType, Integer, Integer, Date)`. The first indicates the type of sensor (i.e.: ACCELX, HUMIDITY, LIGHT, …). The second contains the actual sensor reading. The third is the epoch number of the mote and indicates approximately how long time it has been alive. The last is the timestamp associated to the read value.

Based on the format, the client can create a MoteLimeTemplate and perform an operation. The constructor’s options allow setting the freshness of the data and the operation scope.

**Interaction Between Client and Base Station**

In the base stations, MoteLimeTupleSpace internally exploits two LIME tuple spaces, one holding data from the sensors and one for communicating requests from the clients. It uses two names, `motes` and `config`, to share the content of the two tuple spaces. These tuple spaces are also instantiated in the clients.

Sensor data is retrieved only on demand, what promotes energy saving. When a client issue a request for data, the internal processing of MoteLime TupleSpace first queries the motes tuple space for a match. If there is no match, the base station is informed to query for the required data. It is done by placing a querying tuple in the
config space and registering a reaction on the motes tuple space. It triggers a reaction on the base station that retrieves the data from the sensor and posts it in the motes tuple space what causes the fire of the previously registered reaction and delivers the data to the client.

All base stations run an instance of MoteAgent, which installs the system reaction necessary to the described processing, manages the operation requests, and maintains the freshness of the sensor data.

### 4.9.4.3 Base Station to Mote Interaction

The interaction between base station and motes are done by MoteAgent, as already explained, TOSMoteAccess that asynchronously interacts with MoteAgent to handle requests, replies and all communication with motes, and the other components in the motes.

The main job of the TOSMoteAccess is to translate high-level requests issued by TinyLIME into packets understandable by the motes.

The communication is based on message passing, but with a special care about the energy saving behaviour of the motes. As motes are sleeping a good part of the time, and so, can not receive packets, the base station retries the same package several times. Figure 28 illustrates this communication. In order to avoid duplicate delivery of packages, they are identified by a sequence number in order to motes filter them. As base stations do not sleep, the reply from a mote is just transmitted once. If the base station does not receive a reply within a certain time, it resends the request for the data.

![Figure 28 – Communication between Motes and BaseStation.](image)

### 4.9.4.4 On-Mote Components

In the Figure 29 the software architecture deployed (shaded components) on motes is presented. It is interconnected with TinyOS components (components with white background). The Core module links all components, triggering events and calling parameters. The Timers module controls the epoch and the awake periods. The Filtered Communication module receives all incoming packets, eliminating duplicates based on packet identifiers. Sensor readings, including powering sensors on and off before and after use, are handled by the Sensors Subsystem, which invokes the appropriate TinyOS components. Additional sensors added to the system require modification in this component. The Tuning module handles the setting of mote parameters such as transmission power.
4.10 AWARE Project Middleware

This middleware is a part of a European project called AWARE (Autonomous self-deploying and operating Wireless sensor-actuator networks cooperating with AeRial objEcts) (GIL et al 07). The goal of this middleware is to provide integration of the information gathered by different type of sensors, including WSN and mobile robots.

The AWARE platform consists of two networks, a high bandwidth network (HBN) and a low bandwidth network (LBN). The first is composed of high-end nodes, such as personal computers, cameras and mobile robots, relying in transmission technology based on IEEE 802.3 and IEEE 802.11. The second network is composed of low-end nodes, with very limited computing and data transmitting capabilities. It constitutes the nodes in the WSN. These two networks are connected through gateways that are devices capable to communicate with both networks. The middleware has the goal to make the communication between these two networks transparent. Figure 30 a scenario in the interaction of the two networks can be observed.
The user, which may be a human operator or an application running on a high-end node, describes interest for certain events by defining groups and associating them to channels that denotes the information that he or she or it wants to obtain from the specified group. Groups and channels are defined in terms of attributes, which are numerical values denoting the position of a node or a sensor reading, for example.

The system provides adaptations in order to respond possible changes in the network topology, due to changes in the environment or node failures. The system also provides scalability as new nodes in the network receive the configuration information from nodes already deployed. It gives robustness to the system as one only survival node is enough to transmit the configuration to new added nodes.

4.10.1 Data-Centric Approach and Related Concepts

The data-centric idea of this middleware is related to the definition and management of the groups and channels. In the following the concept of groups and channels is explained with more details to make clear the data-centric approach.

Groups:

One type of group defines some environmental condition of interest defined by the user. Whenever the value read by the sensors of a node meet these conditions, the node belongs to that type of group. A restriction is imposed to ensure that different objects are mapped as separate groups: the communication between nodes that belong to the same group must be possible without passing through nodes that do not belong to that group.

Groups are identified by two numbers: the group type and group identifier. This makes possible different objects of the same type be distinguished. Figure 31 presents an example in which two “fire objects” are distinguished.

![Figure 31 – Operational Scenario Presenting the Distinction Between Two Objects of the Same Type.](image)

Each group has a leader, which is elected by during the creation of the group. Data aggregation inside the group is done by means of use of a routing tree in each node, while the leader makes the final aggregation and sends the final data to the gateway.

Groups can move to follow the area in which the environmental condition of interest occurs. It permits the tracking of objects. If the leader will be out off the area in which the conditions match by the movement of the interested phenomenon or by its own movement, it sends a handover message. Figure 32 presents this situation.
Groups of the same type can also merge or split to follow the movement of conditions of interest. Figure 33 shows a situation in which two groups of the same type merge.

Information about the types of groups defined by the user is epidemically distributed through the network.

**Channels:**

The definition of a channel is based on the information provided by the user about which sensor is to provide the required data, sampling rate, the group type and the group ID to which the channel is associated. Data aggregation is down through the channel. Commands channels are also support, allowing online changes in the configuration.

4.10.2 Implementation Details

The middleware provides a data-centric public-subscribe approach to access the nodes in the WSN. The application layer is responsible to provide information about the node to the middleware. The authors claim that it provides independence of the hardware that supports the node.

The nodes enter in a group when there is a matching of the group conditions. A node joins a group as a member, becoming publisher of any channel to which that group is associated. If there is no group in the vicinity of the node, it creates a group and becomes its leader.

The leader election mechanism avoids conflicts related to duplicated leaders. When a node creates a group it becomes its leader and begins to send leader beacons that let the nodes in the vicinity know about the existence of a group. These beacons are rebroadcasted by all the nodes in the group. Nodes that do not belong to the group do not retransmit the beacons. A leader has a weight that increases as it receives data to
aggregate from other nodes in the group. This weight is used to decide which node will be the leader in the case of more than one leader appear or in case of merging of two groups. The heavier leader prevails. It is achieved by allowing nodes to forward only beacons from the heavier leader and when a leader receives a beacon from a heavier leader, it becomes a member of that leader’s group.

Channels can be associated to more than one group. In order to distinguish information provided by different groups that use the same channel, the data packages include the group ID.
5 THE BIG PICTURE

Based on the requirements put on WSN and the main challenges in the development of adaptable middleware for WSN, this chapter presents a big picture in which the main needs in the area are high-lighted and positioning the analysed proposal face these needs.

The features used as base for the comparison were:

- **Code Mobility**: it evaluates the support of code mobility, both for update and installation of new services in a node;
- **Flexibility**: it evaluates the support for network scalability and support for manage incoming nodes in the network, as well as the manage for the network topology;
- **Node Mobility**: it evaluates the support for mobile nodes in the network;
- **Node Heterogeneity**: it evaluates the capacity of the middleware address the needs of both low-end nodes, with few and constrained resources, as well as more sophisticated sensors, with more powerful computer platforms and advanced resources;
- **Application Knowledge**: it evaluates the ability of the middleware to respond the needs of specific applications or group of applications;
- **Data Fusion**: it evaluates the support for data aggregation and fusion by the nodes that are in the way of data moving from the phenomenon occurrence to the end user that requested the information;
- **QoS**: it evaluates the support for QoS control that can be provided by the middleware.

Evaluating those features, the analysed middleware were graded according to the following levels:

- **4 - Very Good**: provides a promising approach that addresses the challenges related to the concern;
- **3 - Good**: provides a support that partially addresses the challenges related to the concern;
- **2 - About Average**: provides a support that addresses a few of the related problems under concern;
- **1 - Weak**: provides a support that poorly addresses the related concerns;
- **0 - Very Weak**: minimally addresses or does not address at all the related concerns.

Table 2 summarizes the evaluation done.
By the information condensed in Table 1, one can see that there is much to do in the area. There are few grades “4” and “3”, which means that most of the considered features are poorly addressed by existing middleware and that they concentrate in just one feature. As we analysed in Chapter 2, there is a need for solutions that combine efforts in all of those evaluated features in order to respond the adaptation needs that are emerging with the new sophisticated applications of wireless sensor networks.

We call attention for at least two of the analysed features, the “Node Heterogeneity” and “QoS”. The first had low grades with some proposals address partially the problem, but we can identify a great need for research in the area. The second had extremely low grades for almost all the approaches, and one very good grade for MiLAN. We can see that this proposal concentrate it efforts in a QoS solution for WSN and successfully achieves good results, but it is not integrated with the other needs represented by the other features.

The conclusion that can be drawn by this analysis is that there is a need to integrate the support for each of the described feature a common middleware platform in order to offer the required support for new emerging applications.

<table>
<thead>
<tr>
<th></th>
<th>Code Mobility</th>
<th>Flexibility</th>
<th>Node Mobility</th>
<th>Node Heterogeneity</th>
<th>Application Knowledge</th>
<th>Data Fusion</th>
<th>QoS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVIN</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ATLAS</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>AGILIA</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>IMPALA</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SINA</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TinyCubus</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>MILAN</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>SensorWare</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TinyLime</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AWARE</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
6 CONCLUSION

In this report we presented some of the main proposal for the area of adaptable middleware for wireless sensor networks. We first analysed the key challenges in the area, exposing the needs for adaptation and the support that middleware in this domain is required to offer. Afterwards we presented some of the most successful traditional approaches for wireless sensor networks and pointed out some of the problems why they do not address the adaptation needs of the emerging applications. In the following we presented an individual analysis of some of the key approaches in the area, and concluding the report, provided a comparative evaluation considering some key features.

The information given by the research done as base for this report gave us a feeling that there is much to be done in the area of adaptable middleware for wireless sensor networks. The comparison among the analysed proposal pointed out some common problems, such as the lack of support for QoS in the majority of the analysed middleware. Moreover, a concentration in just one of the features was another trend observed. It is a problem as with the presentation of the challenges in Chapter 2, it became clear a need for a solution that manage to address more than one of the features at same time, and not only one of them, because it makes the approach be limited to a certain kind of application or a certain operation scenario or even both.

Emerging sophisticated applications, such as area surveillance systems, require a middleware that combine different supports to their demands for adaptability in one platform. As we could see by the state-of-the-art middleware analysed, this support is no provide yet. More research in the area must be done in order to address the specific needs identified and respond the demands emerging with the new applications and user requirements.
7 REFERENCES


