Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 593 – 600

ISBN 978-83-60810-14-9
ISSN 1896-7094

Real-time Support in Adaptable Middleware for Heterogeneous Sensor Networks

Edison Pignaton de Freitas
IDE – Halmstad University – Sweden / PPGC UFRGS - Brazil
Email: edison.pignaton@hh.se

Marco Aurélio Wehrmeister
PPGC UFRGS – Brazil
Email: mawehrmeister@inf.ufrgs.br

Carlos Eduardo Pereira
PPGC UFRGS - Brazil
Email: cpereira@ece.ufrgs.br

Tony Larsson
IDE – Halmstad University – Sweden
Email: tony.larsson@hh.se

***Abstract* — The use of sensor networks in different kinds of sophisticated applications is emerging due to several advances in sensor/embedded system technologies. However, the integration and coordination of heterogeneous sensors is still a challenge, especially when the target application environment is susceptible to changes. Such systems must adapt themselves in order to fulfil increasing requirements. Especially the handling of real-time requirements is a challenge in this context in which different technologies are applied to build the overall system. Moreover, these changing scenarios require services located at different places during the system runtime. Thus a support for adaptability is needed. Timing and precision requirements play an important role in such scenarios. Besides, QoS management must provide the necessary support to offer the flexibility demanded in such scenarios. In this paper we present the real-time perspective of a middleware that aims at providing the support required by sophisticated heterogeneous sensor network applications. We propose to address the real-time concerns by using the OMG Data Distribution Service for Real-time Systems approach, but with a more flexible approach that fits in the heterogeneous environment in which the proposed middleware is intended to be used. We also present a coordination protocol to support the proposed approach.**

I. Introduction

SENSOR network applications are becoming more complex due to the use different kinds of mobile and sophisticated sensors, which provide more advanced functionalities [1] and are deployed in scenarios where context-awareness is needed [2]. To support those emerging applications, an underlying infrastructure is necessary. The current proposals suggest the use of a middleware, such as TinyDB [3] and COUGAR [4]. The main drawbacks of these state-of-the-art middleware are the following assumptions: (i) the network is composed only by a homogeneous set of basic or very constrained low-end sensors; (ii) the lack of intelligence of such network that compromises the adaptability required facing changing operation conditions, e.g. lack of QoS management and control. Adaptability is a major concern that must be addressed due to: (a) long deployment time of wireless sensor networks may require flexibility in order to accomplish changes in the requirements during usage life time of the network; (b) wireless sensor networks are deployed in highly dynamic environments, implying that applications have to be flexible enough in order to continue being used in these scenarios. In such environments, real-time requirements are especially hard to be met, because of variable operational conditions, and thus there is a need of adaptation of real-time parameters to operational conditions. QoS management must therefore be flexible, allowing renegotiation among nodes during the system runtime [5].

This paper reports a work in progress related to the development of an adaptive middleware to support sophisticated sensor network applications that must adapt its behavior according to changes in the environment and the application demands. We use the concept of multi-agents to provide the reasoning about the network and, besides other things, to decide about time-related requirements and QoS control. This paper focus in the real-time features itself without considering how the multi-agents perform the reasoning. Based on the main real-time concerns that affect heterogeneous sensor networks, we propose the use of mechanisms to address them supported by a coordination protocol. The main contributions provided by this paper are the description of the proposed handling of the outlined real-time concerns by means of the proposed mechanisms and coordination protocol. Besides, the overall middleware proposal consists in a contribution by providing a flexible variant of a middleware for heterogeneous sensor networks based on an OMG standard.

The remaining of the text is organized as follows: Section 2 presents an overview of the proposed middleware. Section 3 discusses some main related real-time issues in middleware for wireless sensor networks. Section 4 presents the proposed approach to support real-time requirements in the proposed middleware. Section 5 presents the coordination protocol that partially supports the proposed approach. In Section 6 some related works are outlined, while Section 7 gives some concluding remarks and directions of the future work.

II. Overview of the Proposed Middleware

The general idea is to develop a flexible middleware that can be used to support applications in heterogeneous sensor networks. In the context of this paper, heterogeneity means that nodes in the network may have different sensing capabilities, computation power, and communication abili-

This work is partially supported by KK Foundation.

ties, running on different hardware and operating system platforms. The main goal is that the proposed middleware fits both low-end and rich sensors. In order to achieve this goal, aspect and component oriented techniques will be used in a way similar to the approach presented in [6][8] and the mobile multi-agents approach [9].

Low-end sensors are those with simple capabilities, such as piezoelectric resistive tilt sensors, needing limited processing support and communication resource capabilities. Rich sensors comprehend powerful devices like radar, visible light cameras or infrared sensors that are supported by moderate to high computing and communication resources. Thus, in order to deal with these very distinct capabilities, the proposed middleware must be lightweight, while being scalable and customizable. For instance, it might handle the node's resource usage in order to assist tasks distribution among different nodes that are capable to accomplish them. Another feature of the middleware is to help provide the quality of the data required by a certain user's demand, such as accuracy and precision. Better results can be achieved by choosing the correct set of sensors to perform measurements and data collection. The mobility characteristic is also related to the heterogeneity addressed by the middleware. Sensor nodes can be static on the ground or can move themselves on the ground or fly over the target area in which the observed phenomenon is occurring. The Fig. 1 graphically represents the idea of the heterogeneity dimensions considered in this work.

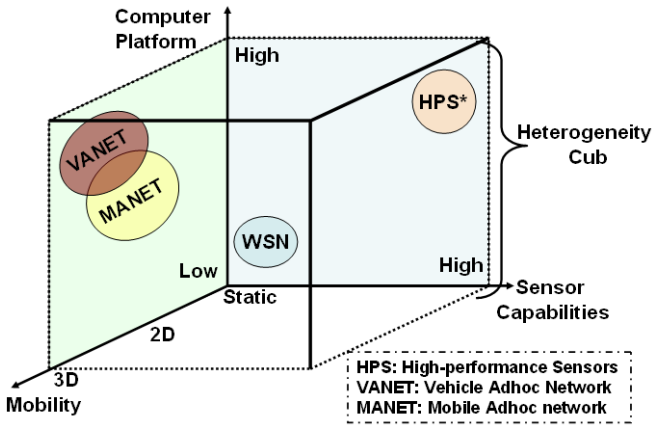

Fig. 1 Heterogeneity Dimensions

The input to the sensor network system, coordinated by the proposed middleware, is seen as a "mission" that the whole network has to accomplish. In order to allow that, a high-level Mission Description Language (MDL) is being formulated based on the C/ATLAS test language [10]. This language will allow the specification of data (at a high level of abstraction) in which the user is interested, including constraints regarding timing and location limits as well as the measurement rate or accuracy desired. This high-level user information will be translated to system parameters, such as QoS related parameters. The proposed language will also allow hierarchical description of the mission goals, with establishment of priorities and other details, for instance, the application of comparison metrics to evaluate how well the mission is being accomplished. The priorities given to missions will also affect QoS parameters. Consequently, the middleware must handle them in order to fit QoS parameters according the established priorities for each task. As an example of usage of the MDL, the user may want to gather data about a certain kind of vehicle that passes in a specific area during a given period, with a high accuracy level. In order to do this, it will specify the target area, a distinguish characteristic of the vehicle (e.g. over a certain weight or with a certain shape), provide the period during which the mission will be performed and how accurate results he or she desire from the system. This high-level information is though translated in internal parameters that will drive the network to perform the mission.

The middleware must perform its actions also in very dynamic and changing scenarios. Thus the set of sensors chosen in the beginning of a mission may not be the most adequate one during the whole mission. For example, an area surveillance system receives the mission to survey an area that may not allow traffic of certain kinds of vehicles. Ground sensors are set to alarm in the presence of unauthorized vehicles. Additionally, Unmanned Aerial Vehicles (UAV) equipped with visible-light cameras is set to fly to the area where a ground sensor has issued an alarm to verify the occurrence. However, a sudden change in the weather, e.g. the area becomes foggy or cloudy, turns the use of a visible-light camera useless. This type of change in the operational conditions must be supported by the middleware, which must be able to choose a better alternative, among the set of available options, for instance by choosing an UAV equipped with an infrared camera instead.

The dynamicity of the operation scenarios may force the adaptation of system's real-time parameters in order to accomplish a certain mission. As an example, certain data forwarding traffic may overload a node in the path from the data gathering points to the final user. Incoming data can experience problems like undesired delay or unpredictable jitter, so solutions as data flow priority assignment and/or use of another node as forwarder may take place. Priorities and choice of alternative paths may further not be static from the start of the system runtime, but can dynamically be changed according the user requirements and changes in the network such as node failures.

The middleware is divided in three parts or layers indicating that they are partly using each other in a specific order. Fig. 2 presents the overview of the layers of the proposed middleware, and a description of each layer is provided as follows.

The bottom layer is called *Infrastructure Layer*, which is responsible for the interaction with the underlying operating system and for the management of the sensor node resources, such as available communication capacities, remaining energy, and sensing capabilities. This layer also coordinates the resource sharing based on application needs passed through the upper layers. Services provided by upper layers may need some resource sharing support, which is encapsulated in the infrastructure layer. As an application uses such a service, the corresponding layer asks for the infrastructure layer to manage the access control to the required resources.

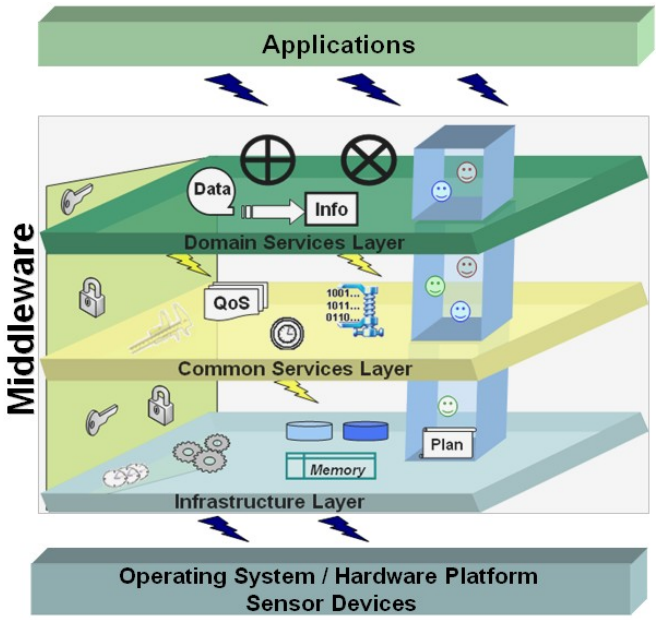

Fig. 2 Overview of the Middleware Layers

The intermediate layer is called *Common Services Layer*, which provides services that are common to different kinds of applications, such as QoS negotiation and control, quality of data assurance, data compression and the handling of real-time requirements, including storing of parameters. Other concerns such as deadline expiration alarms, timeouts for data transmissions, number of retries and delivery failure announcements, resource reservation negotiation among applications (based on priorities and operation conditions), dynamic bindings, synchronous and asynchronous concurrent requests are also handled by this layer.

The top layer is called *Domain-Services Layer* and has the goal to support domain specific needs, such as data fusion support and specific data semantic support to allow the production of application-related information from raw data processing. Fuzzy classifiers, special kinds of mathematical filters (e.g. like Kalman filter) and functions that can be reused by applications of the same domain will be found in this layer.

Multiple applications can run concurrently in the network. The middleware handles resource sharing and provides data sharing among applications that need the same type of data, allowing a better energy use in resource constrained nodes. In powerful nodes, with more energy available, the middleware can provide more complex services aiming at the handling of rich data, such as those related to image processing, and pattern matching. This also means that such nodes can take some of the burden from meager nodes.

"Smile faces" in the Fig. 2 represent agents that can provide specific services in a certain node at a certain moment of the system runtime. A special region (called *agents-space*) links them throughout the layers, allowing the exchange of information. The *Infrastructure Layer*, in the bottom, has just one agent, which is responsible for planning and reasoning activities. Interested readers can find more information about the use of agents in our middleware in [9].

Concerns that affect elements in more than one layer of the middleware, such as security, are represented as cross-layer features. In Fig. 2, the "locks" and "keys" in the left-side plan represent this idea for the security concern example. These crosscutting concerns will be addressed in our middleware with the aspect-oriented approach presented in [6] and [7]. Dark lightning bolts represent the communication activities between applications and the middleware in the upper part of the Fig. 2 and between the middleware and underlying operating system and hardware (including sensor devices) in the lower part of the Fig. 2. Light lightning bolts in the middle of Fig. 2 represent communication among internal elements of the middleware.

III. Key Features for the Wireless Sensor Network Middleware

The proposed middleware is intended to be used in dynamic harsh environments. Changes in this kind of scenario occur frequently, requiring system adaptability. However, it is not useful to provide adaptability without guaranties that the desired data will be delivered in time. So, the timing related concerns about the network consist also in an adaptable dimension that the middleware must take into account in order to successfully provide the necessary support to applications running within those scenarios. With this thought in mind, some key features that the middleware for wireless sensor networks must take into account are presented [11] [12]:

- **Avoid Single Point of Failure:** no single node must centralize any type of registration or submission service for the entire network, as in broker-based architectures. This concern is related not only with the possible node fail, but also with node overload and incapacity to respond the demands of all clients;
- **Dynamicity in System Structure and Network Topology:** as operation conditions change, new nodes can come in and out. Moreover, different services can be required in different places at different times;
- **Reduce Data Communication:** As large amounts of data may need to be exchanged among nodes, the bandwidth must be carefully used to avoid congestions and flooding;
- **Meet Applications Requirements:** Applications must meet timing requirements, based on their users' needs for data, and on the conditions under which they want the data. These requirements may change during runtime, what requires a fine tuned QoS management and control;
- **Real-time Requirements:** The middleware must provide the specification of real-time requirements. At the same time, it must handle the translation of these requirements into specific QoS parameters, in order to accomplish the modification of this high-level requirements;

- **Fault-Tolerance:** Nodes may fail, links can go down and network errors occur, but the application that requests data cannot wait indefinitely. Mechanisms of errors reporting must be provided, as well as alternative resources may be offered. Activities such as time-bounded activities monitoring, delays and deadlines management must be handled in order to provide dependable support.

These above highlighted issues crosscut several concerns, i.e. they are intertwined with several characteristics. For example, QoS values, such as deadlines and bounded time actions/delays, impact in the management of communication and fault-tolerance. The resulting complexity indicates a need to use proper abstractions in order to deal successfully with these concerns.

IV. Addressing Key Issues of Sensor Networks with the Proposed Middleware

The proposed middleware is inspired in the Data Distribution Service for Real-time Systems (DSS) specification, standardized by OMG [13]. The proposed approach is based on the publish-subscribe paradigm. Some nodes publish their capabilities and the offered data, while others subscribe to data, in which they are interested.

Although being inspired on the OMG DSS standard, the middleware does not follow the whole specification. As it is intended to fit both low-end nodes (based on simple and constrained platforms) and sophisticated ones, it must not only be lightweight but also provide capabilities for customizations in order to deal with the needs of the sophisticated sensors. Consequently, the middleware uses a minimalist approach, keeping it as simple as possible in each node. It may be considered as a "Lightweight Flexible DSS-based Middleware", in which the handling of some features will be presented in a soft version, such as the handling related to topics described in the ***Domain Module*** section of the DSS specification [13]. This soft version will simply data structure and check mechanisms that will be deployed middleware instance for low-end nodes, for instance.

Additionally, features can be included or modified in the middleware by using adaptation and extension mechanisms, such as the inclusion of additional components, as well as the weaving of specific behaviors by aspects. As an example of more explicitly of how the aspects will be used, the handling of QoS policies, which affect different elements in the system, can be concentrated in a aspect avoiding the spread of this concern over different classes, and like this, making it easier to promote maintainability and evolution of those policies. In the DSS specification, several classes (e.g. , `DomainParticipantFactory`, `Topic`, and `DomainParticipant`) have parameters and methods to handle the QoS policy concern. Using aspects, the behaviors encapsulated in those methods spread over several classes can be concentrated in one aspect that weaves classes in the system accordingly the policy needs. Other proposed adaptations in the DSS original specification are: (1) the simplification of the status model, which will consider a subset of the original proposed object inheritance tree presented in section dedicated to the details about communication status in [13]; (2) adaptation of the cache model in order to add the cache usage proposal that will presented further in this section.

The following subsections present how the proposed middleware will address the needs presented in Section 3.

A. Flexibility

The middleware provide full control of the communication, it does not use underlying control mechanisms available in the nodes' network layer. Instead, it provides its own communication control. It means that all parameters related to communication are controlled by the middleware, using only basic connectionless communication services offered by the nodes' network layer. The middleware handles parameters like number of retries, message priority, memory usage for buffering and timing. This control over communication provides more flexibility to manage the messages exchanged by each node, with direct impact in the reduction of latency. Moreover, it gives a finer grained control if compared with the simple use of the communication primitives offered by the native nodes' operating system network services.

B. Dynamicity

Using the publish-subscribe paradigm, when a node gets into the network, its services are announced and the interested nodes subscribe for them. This eliminates the need for a dedicated server node that centralizes the available services in the network. Additionally, it reduces latency in acquiring data because there is no intermediary node between the data producer and the consumer.

C. Minimum Message Exchange

Using the publish-subscribe paradigm by itself already reduces the number of exchanged messages, due to the elimination of intermediate nodes, such as request-brokers. However, the use of bandwidth for control messages still exists. It can be reduced with the use of a smart protocol to avoid unnecessary messages exchange, as explained with details in the following section. The protocol is called CUME (Cut Unnecessary Messages Exchange).

D. Multicast Communication

The middleware use a multicast communication to reach selected destination nodes. For instance, the publisher sends its data only to the nodes that subscribed to it. This type of communication affect positively the latency and throughput, as data is sent at the same time to several nodes without unnecessary broadcast and without delays that would occur if a unicast communication was used; since then the publisher node would have to resend the same data several times, one for each subscriber. A negative-acknowledgement (NACK) strategy is adopted in order to reduce acknowledgement messages in the network. However, very sensible data may require a positive acknowledgement in order to assure its delivery. To address this need, positive acknowledgement is also available and can be used when required. This acknowledgment strategy is also part of CUME.

E. Network Resources Usage Control

The control of the use of the communication media and transmission buffers are crucial in order to improve the over-

all system performance. The middleware perform this task by taking into account two factors: (i) the priority associated to each application; and (ii) the resource sharing policy adopted in the system. There are three available resource sharing policies:

- **Fair Sharing:** the priorities are not considered and thus all applications have the same right to use the resources in a round-robin scheme, which is organized in a incoming FIFO queue;
- **Soft Priority Sorted** : the priorities are taken in account. However, if a higher priority application needs to use a resource already used by a lower priority one, it must wait until the later release the resource. Due to its higher priority, it will get access to the resource before other applications, which may be waiting for the resource;
- **Mandatory Priority** : higher priority applications can preempt lower priority applications in order to access the desired resources.

F. QoS Control

The QoS control is done through a contract between the data provider and the data requester. When a node publishes a data service, it informs also the QoS level that it is capable to offer. Nodes interested in the published data service those accept the offered QoS level, subscribe to the service. However, if a node is interested in the data but does not agree with the offered QoS, it has two alternatives:

- If the application that is requiring the data has a priority lower than the others using the same service, it looks for another data provider;
- If its priority is higher than other applications, it negotiates with the data provider node, in order to obtain the desired QoS in spite of the bad consequences that it may imply to other lower priority applications.

As an example, a node may provide a certain type of data (D-1) at each 10 milliseconds and another type of data (D-2) at each 25 milliseconds. The first, D-1, has two subscribers and the second one, D-2, just one. A forth node wants to receive D-2, but at each 10 milliseconds. The data provider node does not have the available resources in terms of processing and communication power to deliver both data at each 10 milliseconds. It can deliver just one type at the desired rate. If the forth node has an application with a priority higher than the applications running in the other nodes, it will negotiate with the provider and, if it is feasible, the provider will change its behavior to accomplish the need of the requesting node with the highest priority. If it is not feasible, the requesting node will look for another provider.

G. Use of Cached Values

Some measurements aim to gather information about changes in values of certain observed phenomena. The use of cache in both data providers and requesters may avoid unnecessary data communication. When the measurement device gathers a new value, the data provider updates its own cache and publishes the new value updating its subscribers. If the size of the data is large and requires several packets to be transmitted, a differential value can be send instead of the whole data value, for instance, using just one package. This differential value will be used to update the current value stored in cache. The use of this option is arranged in advance at the time when the nodes are negotiating the QoS contract.

H. Fault Tolerance

In order to support the use of strategies like cached values and to detect node failures, fault tolerance mechanisms must be provided. A heartbeat mechanism is used when a node does not have any data to send. Thus it broadcasts a message in order to inform other nodes that it is still alive. It works well for nodes in the range, but for those not in the range and that are interested in the provider, directed heartbeats must be sent. The periodicity of the heartbeat sending is a configurable parameter. Another fault-tolerance support is a list of possible backup service providers. When a node that provides a certain kind of data service stops working, nodes interested in that data look for another node at the backup list that provides the missing data.

I. Network Partitioning

Network partitioning can occur intentionally or by uncontrolled conditions. The first one occurs when groups of nodes have much communication among each other and little outside the group. Thus, they form a cluster in order to minimize the communication with outside nodes. A cluster-head, which is responsible for communications with clusters outside nodes, is elected. This election is based in the actual status of the involved nodes, concerning remaining resources and quality of the communication link. The use of a cluster-head does not characterize a single point of failure because any node in the cluster can be elected. If the current cluster-head fails, a new one is elected. The cluster-head formation is used only if the QoS requirements are met. For instance, if the presence of the intermediary communication with the cluster-head impact latency, a communicating node can leave the cluster. If it has a specific need for a data that is not interesting to the other nodes (or have different QoS requirements) it can make a contract directly with the provider of the required data, without passing the cluster-head. This way the cluster of nodes is not rigid, but flexible.

Uncontrolled conditions, such as node failures, communication obstacles and interference can also promote network partitioning. This kind of partitioning is not desirable and planned such as mission related clustering, and to handle this issue, isolated nodes store as much data as possible to send when the link be reestablished, what can be done by the deployment of new nodes, by the mobile nodes that come into the area or by the disappearance of the obstacles or interferences that caused the partitioning.

J. Data Segregation

There are two kinds of data exchanged among nodes in the network: control data and application data. Control data is small and may not experience latency or unexpected delays to achieve its destination. So, control data is segregated from application data by receiving higher priority to be forwarded. On the other hand, there are several kinds of appli-

cation data, e.g. simple values (integers and floats), video stream and character string. In spite of this sort of data have a priority lower than control data; they must fulfill the QoS requirements of the application. Moreover, jitter is also reduced by the segregation, because segregated data follows different buffers.

K. Synchronous and Asynchronous Calls

The middleware is intended to support both synchronous and asynchronous calls. Synchronous calls are bounded in time in order to avoid unpredictable waiting periods by the caller applications. The waiting time and number of retries are configurable. In case of the expiration of the waiting time (timeout), or if the number of retries is reached, specific handling mechanisms can be triggered. Asynchronous calls are also provided and they are used, among for other proposals, to support the handling of asynchronous events.

V. CUME Protocol

CUME (Cut Unnecessary Message Exchange) is a protocol that aims at minimizing the number of control messages exchanged among nodes, and additionally to optimize the use of data messages.

The publisher-subscriber approach proposed is slightly different from the state-of-the-art middleware available. Instead of announcing its service when coming into the network, a node does that in different manners depending on the situation, on the type of the node and data provided.

For instance, let's consider moving sensors embedded in UAVs, which fly in a cluster formation to accomplish a mission in a certain place (passing through several way-points). The group of UAVs can find another UAV flying in the same direction, which we call an alone-UAV. In this case, the protocol will perform the following: (i) exchange nodes' destination information; (ii) exchange capabilities and mission details information; and (iii) negotiate the use of the nodes' resources if it is the case. In the following each of these phases are explained with more details.

The alone-UAV just tells next way-point to the clusterhead of the UAV-cluster. Then the later sends a message with the next way-point destination to the cluster. If the destinations are the same, the alone-UAV joins the group without extra confirmation messages. There is no acknowledgement message, if one of these sent messages was lost, the other node that was expecting to receive the message send a NACK requiring retransmission. The waiting time for sending the NACK is estimated by the distance between the nodes, and then, a timeout is set in accordance.

Being in the cluster, the alone-UAV can use short range communication to announce its services and the mission that is supposed to accomplish. If there are common interests among the alone-UAV and the UAV-cluster, they cooperate. If there are not, they analyze the capabilities of each other and recognize if any of them is needed for its own mission and if it lacks this capability. Based on the established priorities and the mission conditions, they will decide which mission is more important and the resources will be allocated to that mission according to the priorities. This is an example in which unnecessary long-range communications were avoided.

In the case of different destinations, they will perform the exchange of the capabilities information anyway using long-range communication to make the same reasoning described above.

The CUME protocol in addition uses caches in the nodes to avoid unnecessary messages exchange. If a publisher has two subscribers of a certain data; one that requires the data at each 10 milliseconds (called node A) and another at each 20 milliseconds (called node B). After the data provider sends the first value for each, it stores the value in its local cache. Nodes A and B receive the data, giving it to the requesting applications and storing it in their caches. After 10 milliseconds, the provider compares the measured value with the one stored in its cache and, if it is different, sends it to node A, updating its cache. However, if the value is the same, it just sends a heartbeat to node A, which understands that the value is the same and gives the value in its cache to the requesting application. In the next 10 milliseconds, the same occurs, but the update or the heartbeat will be sent to both requesting nodes A and B. In this situation there is no acknowledgement for each message. If an expected message is lost, a NACK is sent from the consumer nodes to the data provider. In order to avoid a NACK storm in the case of several requesting nodes miss the sent data, a random timer is set in each node that loosed the message to send the NACK. The value that is set in the random timer is chosen in a range according to the established QoS requirements. When a value must be updated, it is possible that just the difference between the new and the old value is sent. This is also a parameter exchanged during the negotiation.

Caching negotiation is also part of the CUME protocol. It is done by an exchange of control information that will represent the agreement about the freshness time of the cached values, and the accepted delay to receive the refresh of the cached data or a new value. Besides, the cache space is not a fixed amount of memory but it is negotiated. These parameters can be renegotiated according to the needs and/or changes in the environment and operation conditions. Threshold values for renegotiations are established in the first negotiation round, in order to avoid unnecessary exchange of control messages due to small changes, which do not represent a real need for renegotiation. The parameters will depend on the application and the type of mission, thus a flexible set of parameters can be used according to the individual needs presented in different situations.

The CUME protocol is also responsible for monitoring the rate of messages forwarded by the nodes. If the node's throughput is arriving to its limit, the node informs the sending nodes to use different nodes to forward their messages, according to the priorities of the applications that require the communication. The limit of bandwidth usage is also a configurable parameter that can be adapted according to the operational conditions, such as interferences or amount of collisions.

VI. Related Works

MiLAN [14] (Middleware Linking Applications and Network) is an adaptable middleware that explores the concept of proactively adaptation, in order to respond the needs in terms of QoS imposed by changes in the operational environment. The authors claim that a major drawback in the most of existing middleware for sensor networks is the fact that they just provide reactive adaptation. In dynamic environments as those in which wireless sensor networks are deployed, this behaviour does not cover the real needs for adaptation required by running applications. MiLAN allows the specification of the required QoS for data, adjusting the network to increase its lifetime, by efficiently using energy, in the same time that the quality needs are still meet. The major difference in relation to our approach is that MiLAN does not provide a customization mechanism to enrich the middleware features and support more sophisticated sensor nodes like those carried by UAVs. This difference points to a drawback indicating that MiLAN is not prepared to manage high intense data flow generated by traffic of images or video streams.

Atlas [15] is an architecture for sensor network based on 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 Atlas middleware is based on the OSGi Service Platform Specification [16], which is a framework that provides a runtime environment for dynamic and transient service modules called Bundles. This middleware offers some support to real-time, but as a secondary feature derived mainly from the adaptable service discovery feature and Bundles specification. However, this support is too simple and do not promote the desired control of real-time parameters as we proposed in our approach. Furthermore, the Atlas middleware does not fit into low-end nodes with constrained resources.

Real-time CORBA (RT-CORBA) [17] is a successful middleware that provides real-time support for distributed applications. It is based on the CORBA standard [18], which introduces a priority mechanism to map native operating system priorities into remote nodes priorities. It also provides predictability by managing resource allocations according to the established priorities. The use of thread pools prevent the unbounded blocking periods based on the resource usage by the lower priority applications, and priority inversion. RT-CORBA fits well in sophisticated sensors nodes with a rich computing platform, but it is too heavy to run in low-end nodes. In our approach instead, the proposal is to address the real-time needs in both low-end and rich sensor nodes with different computing platforms and available resources.

Quality Objects (QuO) [19] proposes the addition of a QoS adaptive layer on existing middleware, such as RT-CORBA. It provides means of specify QoS requirements, monitor and control the provided QoS, and also adapt the middleware behavior according to the QoS variations that may occur during runtime. This proposal presents an interesting approach to support those operations using: (i) *contracts*, which encloses the QoS requirements; (ii) *delegates*, which are proxies that can be inserted into the path of object interactions transparently to weave the QoS awareness and adaptive code; and (iii) *system condition objects*, which provide consistent interfaces to infrastructure mechanisms. However, as this framework relies on an existing middleware such as RT-CORBA, it has the same drawback indicated above, i.e. it cannot be used in low-end nodes.

VII. Conclusion and Future Works

This paper presents the real-time support offered by an adaptable middleware for heterogeneous wireless sensor networks. Real-time concerns, which affect a network composed by heterogeneous sensor nodes and impose difficulties on the handling of them, are handle through the proposed middleware that fits both rich sophisticated nodes and low-end constrained nodes. Our proposal presents a protocol to cope with the unnecessary control message exchange, and use different mechanisms to address real-time issues. Besides bandwidth savings, it has the side effect to increase the security against eavesdroppers, by diminishing the number of exchanged messages. The use of mechanisms like caching and differential updating is also provided in order to diminish the bandwidth usage.

Related works in the area do not address both types of sensor nodes. The majority of middleware for sensor networks consider a homogeneous network composed by low-end nodes, producing very simple data, like the approach presented by MiLAN. In the other extreme there are middleware that consider a network composed by powerful sensor nodes that, in some cases, do not even have any concerns about energy consumption, as the assumption presented by Atlas. Conversely, the proposed middleware addresses both low-end and rich sensor nodes, using mechanisms that provide support to both simple and sophisticate data.

As future works we are refining the mechanisms of the CUME protocol in order to incorporate different strategies to provide a better response to problems like jitter. The coordination mechanisms are also under development. In this paper we gave an overview of the type of coordination that it will be provided, as presented in example of the UAV-cluster meeting the alone UAV. However, we are working to provide similar kinds of coordination in other situations and with different kinds of sensor nodes. Moreover, we still have to simulate a complete case study scenario and assess the measurements in order to validate our assumptions before the final implementation; and also concerning the implementation, proposals of use composed link metrics in the message forwarding decisions (routing) [20] and probabilistic tendencies for the cluster-head election [21] are being considered.

Acknowledgment

E. P. Freitas thanks the Brazilian Army for the grant given to him to follow the PhD program in Embedded Real-time Systems in Halmstad University in Sweden in cooperation with UFRGS in Brazil.

References

[1] D. Culler, D. Estrin, and M. Srivastava, “Overview of sensor networks” , *IEEE Computer* , vol. 37, no. 8, pp. 41–49, 2004.

[2] K. Henricksen and J. Indulska. “A software engineering framework for context-aware pervasive computing”, *In 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom),* pages 77–86. IEEE Computer Society, March 2004.

[3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. “TinyDB: An acquisitional query processing system for sensor networks”, *ACM Transactions on Database Systems* , 30(1):122–173, 2005.

[4] P. Bonnet, J. E. Gehrke, and P. Seshadri. “Towards sensor database systems”, *In 2nd International Conference on Mobile Data Management (MDM)* , volume 1987 of Lecture Notes, 2001.

[5] V. Liberatore. “Implementation challenges in real-time middleware for distributed autonomous systems”, In *Prof of Second IEEE International Conference on Space Mission Challenges for Information Technology, 2006. (SMC-IT 2006)*.

[6] E. P. Freitas , M. A. Wehrmeister, C. E. Pereira, F. R. Wagner, E. T. Silva Jr., F. C. Carvalho. “DERAF: A High-Level Aspects Framework for Distributed Embedded Real-Time Systems Design”. *In: Proc. of 10th International Workshop on Early Aspects* , Springer, 2007, pp. 55-74.

[7] Wehrmeister, M.A., Freitas, E.P., Pereira, C.E., Wagner, F.R. “Applying Aspect-Orientation Concepts in the Model-Driven Design of Distributed Embedded Real-Time Systems”. *In: Proc. of 10th IEEE International Symposium on Object/component/serrvice-oriented Real-time Distributed Computing (ISORC'07), Springer* , 2007, pp. 221-230.

[8] A. Tesanovic, et al. “Aspects and Components in Real-Time System Development: Towards Reconfigurable and Reusable Sofftware”, *Journal of Embedded Computing*, IOS Press, v.1, n.1, 2005.

[9] E. P. Freitas, P. Söderstam, W. O. Morais, C. E . Pereira, T. Larsson. “Adaptable Middleware for Heterogeneous Wireless Sensor Networks”, *In Proc. 10 th European Agent Systems Summer School (EASSS08),* 2008. pp.17-24.

[10] IEEE Std 716-1995, 1995. IEEE standard test language for all systems-Common/Abbreviated Test Language for All Systems (C/ATLAS), IEEE, Inc.

[11] K. Romer, O. Kasten, and F. Mattern, “Middleware challenges for wireless sensor networks,” *ACM SIGMOBILE Mobile Communication and Communications Review* , vol. 6, no. 2, 2002.

[12] I. F. Akyildiz and W. Su and Y. Sankarasubramaniam and E. Cayirci, “Wireless Sensor Networks: A Survey”, IEEE Computer, vol. 38, no. 4, pages 393-422, Mar. 2002.

[13] Object Management Group (OMG). Distribution Service for Real-time Systems (DSS) Specification. Version 1.2. January 2007.

[14] W. Heinzelman, A. Murphy, H. Carvalho and M. Perillo,"Middleware to Support Sensor Network Applications," *IEEE Network Magazine Special Issue*. Jan. 2004.

[15] J. King, R. Bose, Hen-I Yang; S. Pickles, A. Helal. Atlas: A Service-Oriented Sensor Platform: Hardware and Middleware to Enable Programmable Pervasive Spaces. In: *Proc of 31st IEEE Conference on Local Computer Networks* , 2006. pp. 630 - 638.

[16] OSGi Alliance. OSGi Service Platform, Core Specification, Release 4, Version 4.1. May 2007.

[17] R. E. Schantz, J. P. Loyall, D. C. Schmidt, C. Rodrigues, Y. Krishnamurthy, and I. Pyarali. “Flexible and Adaptive QoS Control for Distributed Real-time and Embedded Middleware”, *In Proc. of 4th IFIP/ ACM/USENIX International Conference on Distributed Systems Platforms* , *Springer* , 2003.

[18] Object Management Group (OMG). Common Object Request Broker Architecture: Core Specification. Version 3.0.3. March 2004.

[19] R. Vanegas, J. Zinky, J. Loyall, D. Karr, R. Schantz, and D. Bakken, “QuO's Runtime Support for Quality of Service in Distributed Objects", *In Proc. of Middleware 98* , the IFIP International Conference on Distributed Systems Platform and Open Distributed Processing, Sept 1998.

[20] Heimfarth, T., Janacik, P. Cross-layer Architecture of a Distributed OS for Ad Hoc Networks. In: Proceedings of the International Conference on Autonomic and Autonomous Systems, 2006. ICAS '06. pp. 52- 52, 2006,

[21] Heimfarth, T., Janacik, P., Rammig, F. J. Self-Organizing Resource-Aware Clustering for Ad Hoc Networks. In: Proceedings of the 5th IFIP Workshop on Software Technologies for Future Embedded & Ubiquitous Systems (SEUS 2007), Santorini Island, Greece, Mai 2007