Configuration and Data Processing over a Heterogeneous Wireless Sensor Networks

José Cecílio, Pedro Furtado

University of Coimbra,
Coimbra, Portugal
jcecilio@dei.uc.pt, pnf@dei.uc.pt

Abstract— **Whereas software adaptation has been identified as an effective approach for addressing context-aware applications, the existing work on WSNs fails to support context-awareness and mostly focuses on developing techniques to reprogram the whole sensor node rather than reconfiguring a particular portion of the sensor application software. Therefore, enabling adaptively in the higher layers of the network architecture such as the middleware and application layers, beside the consideration in the lower layers, becomes of high importance. In this paper we propose an approach to hide heterogeneity and offer a single common configuration and processing component for all nodes of that heterogeneous system. This advances the current state-of-the-art, by providing a lego-like model whereby a single simple but powerful component is deployed in any node regardless of its underlying differences and the system is able to remotely configure and process data in any node in a most flexible way, since every node has the same uniform API, processing and access functionalities.**

Keywords - Heterogeneity, Middleware, Distributed systems, wireless sensor networks.

I. INTRODUCTION

Wireless Sensor Network (WSN) embedded device nodes such as the TelosB have limited amounts of memory, processing power and energy, therefore they typically run small OSs for embedded devices (e.g. tinyOS, ContikiOS). They are very different platforms from PC nodes, and frequently they do not run an IP communication stack either. The devices communicate wirelessly and one or more is connected to the internet through the USB adapter and a serial read/write driver as interface, with some form of gateway software above that.

Current solutions for deploying the heterogeneous systems that include WSN nodes and the rest of the world involve either programming every detail of processing and communication by hand, both within the WSN and outside of it, or to use some middleware that works as a single black-box for the whole system (and must be ported to other systems as a whole). The state-of-the-art middleware approaches fail to consider the infrastructure as a heterogeneous distributed system that should have uniform node components over heterogeneous physical nodes. We propose WSN-LightProcessor, a model to enable such vision. By considering a single node component that can be installed in any node, including nodes outside of the WSN, we ensure that all nodes will have at least a uniform configuration interface (API), important remote configuration and processing capabilities without any further programming or gluing together. As an immediate advantage of our proposal, a control station and indeed any node outside of the WSN will have at least the same configuration and processing capabilities and the same interface as the remaining nodes without any custom programming. More generically, in a heterogeneous deployment with different types of sensor devices, control stations and sub-networks, the approach offers easy and immediate homogeneity over heterogeneous infrastructures.

The paper is organized as follows: section 2 discusses related work. Section 3 discusses the architecture of WSN-LightProcessor, then section 4 is on configuration and data processing model. Section 5 presents experimental results and section 6 concludes the paper.

II. RELATED WORK

The main issues raised by WSN programming platforms are the connection to the outside world and the approach followed by the middleware itself. In this section we review both issues.

Interconnecting WSNs to the internet is a subject that attracts a lot of attention of the research community in recent years. At the lower levels, there are some different ways to interconnect TCP/IP networks with sensor networks. In this section we review three main types: using IP protocols, bridges or gateways.

Using IP protocol, which assumes that sensor nodes are powerful enough to run a u-IP protocol [9]: by running u-IP protocol in WSNs, Internet users can access any sensor node by using IP address. But, the assumption that sensor nodes always have enough resources to support the overhead that is brought by the IP protocol stack is not feasible. Different WSNs and embedded devices may have different requirements on this issue.

Using a bridge: In [10] authors proposed VIP Bridge to connect heterogeneous WSNs with IP based networks. Packets that come from one side will be translated into corresponding packet formats and sent to another side by this VIP Bridge. In this research different translators are used for

translating different packet formats, e.g. ZigBee and Bluetooth, into IPv6 protocol packet format.

Using a gateway: The most common approach for connecting sensor networks with an external network is using application-level gateway, e.g. GSN [11]. Different protocols in both side networks are translated in the application layer. In GSN, authors use wrappers to translate different protocols packet formats into unified data format. The main role of gateway is to relay packets to different networks. But due to the lack of the uniform protocol standard of sensor networks, there is no universal architecture suitable for all types of different sensor networks [10]. When the users want to visit other types of sensor networks, they still need to redesign a new gateway. Thus, a gateway-based approach solves the problem of interconnecting certain types of sensor networks with TCP/IP networks.

These solutions can be used to interconnect any WSN with backend servers, and in particular the gateway approach does not assume anything about what network layer code needs to run on sensor nodes. Our proposal includes a gateway component for interconnection at the network level when the WSN network does not support IP directly.

Besides these works, there are other works that address middleware with API's for sensor networks that are also related with our work, we review some next. The main improvement in comparison to existing middleware is our view of a single uniform component with the functionality and interfaces for remote configuration and operation in any node including control station(s), providing simplicity, uniformity and flexibility in where the operations are to be configured.

The work in [3] presents a generic API for sensor networks, focusing on flexibility, extendibility and expressiveness. For instance, retrieving sensor readings depending on location (not just sensor ID) is supported. Similarly, [4] [5] present a very generic view on sensor networks and focus on expressiveness and abstraction. Another option for one sensor network is to use a database oriented interface based on systems like TinyDB [6].

JSR 256 [7] is an API for controlling sensors in mobile devices. It is primarily intended to control sensors inside a device, and does not address remote configuration as our proposal.

None of these works handles the issue of uniform distributed configuration with heterogeneity that we are solving in this paper.

III. WSN-LightProcessor System Architecture

As devices of WSN have limited computation capabilities and may connect with different sensors and actuators, manual configuration/reconfiguration (by programming) is infeasible if a large number of sensor nodes are used. Based on web services, we introduce a plug-and-play approach, that allows configuring/reconfiguring any node in the network.

We designed the WSN-LightProcessor to offer the configuration and execution interface for the wireless sensor network.

Every node is referenceable and configurable from a single system configuration component. This allows every node, including a control station outside the WSN, to have the same configuration and processing capabilities readily available without any customized programming – e.g. detection of thresholds and alarms, closed control supervision code, statistic reports to be computed periodically from the signals. This way the programmer configures processing of a complete WSN application, including what to do with the data outside the WSN.

The WSN-LightProcessor node component depicted in Figure 1 has a unique address and the following modules:

- Sensors and actuators, not depicted in the figure (existing in sensor and actuator nodes);
- A Node Processor, which implements node operations. Node operations include data processing and any other operations that may be defined for a node. Data Processing concerns computations and actuations over the signal data that come to the component from other nodes or from sensors;
- A Config module, which configures the component - data processor, sense and act, I/O Adapter communication;
- An API, which defines the external configuration interface that the WSN-LightProcessor component offers.
- I/O Adapter, which implements a protocol to exchange data and commands with other WSN-LightProcessor components. Remote configuration is based on API calls through the I/O Adapter.

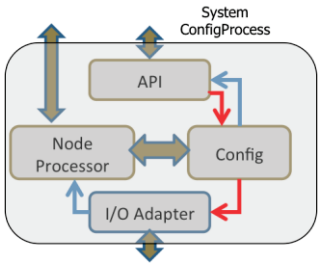

Figure 1 - WSN-LightProcessor Component

The processing part of the WSN-LightProcessor component implements a (simple) data processor that is configured by the configuration part. A complete working system is built based on deploying WSN-LightProcessor components on all nodes, and a system configuration component which will run on a controlling station.

IV. CONFIGURATION AND DATA PROCESSING MODEL

WSN-LightProcessor configuration model offers an API that must provide the set of functionalities required by applications to interact and configure the nodes remotely for operation in the distributed network. We list next some of configuration capabilities that were useful in our project context, but other configuration and operation functionalities should be present depending on the target context for the system or the amount of functionality that is desired. It is also possible to define more powerful APIs and consequently larger WSN-LightProcessor components in more computational powerful nodes (e.g. PCs), but an important subset of basic functionalities should be present in every node. The basic list for our prototype is:

- Activate / deactivate nodes;

```
Node.activate(Zone1SensorNodes);

Node.deactivate(Zone1SensorNodes);
```

- Activate / deactivate sensors and actuators connected to each node;

```
Node.Sensor.activate(Zone1SensorNodes,
                TEMPERATURE);

Node.Sensor.deactivate(Zone1SensorNodes,
                TEMPERATURE);
```

- Gather sensed value data at different frequencies;

```
Node.setSendingRate(Zone1SensorNodes,
                5s);
```

- Request node status;

```
Node.getStatus(Zone1SensorNodes);
```

- Define filters;
  In some environments, a smoothing filter is needed to reduce the noise. The system offers a primitive that allows defining and applying a filter. The filter consists on moving windows that collect a specific number of samples, defined by the user, and compute the average of the samples.

```
Node.createFilter("Zone1SensorNodesFilter",
        Zone1SensorNodes,
        5 );
```

- Define alarms, actions based on conditions;
  As our focus is on creating a configuration interface for industrial contexts, our API includes functionalities to create, drop, start and stop alarms and actions based on conditions that are introduced by users.
  Actions or Emergency actions are used to actuate in a node or set of nodes when a specific condition occurs. The condition is defined the same way for alarms or actions.

```
Alarm.createAlarm(Zone1SensorNodes,
        "NodeTemperatureAlarm",
        temperatureValue,
        CONDITION( TEMPERATURE.VALUE, >, 5)
        );
```

The WSN-LightProcessor implements data processing over signals using a stream processing model. In this model sensor signals are inserted into streams, from then on operations specify manipulations over streams. The creation of streams is through a call to an API method as exemplified next:

```
Node.createStream(Zone1SensorNodes,
        "temperatureStreamfromZone1SensorNodes",
        1s,
        {(PRESSURE, VALUE)}
);
```

A control station can receive signals from the sensors and determine an alarm when temperature rises above a specified threshold:

```
Node.createAlarm(controlSation,
        "ServerTemperatureAlarm",
        temperatureStreamfromZone1SensorNode,
        CONDITION( TEMPERATURE.VALUE, >, 27)
        );
```

Due to the uniform approach of WSN-LightProcessor and remote configuration capabilities, the same command can be submitted for configuration of any node, by simply changing the address field (controlStation and Zone1SensorNodes are node referencing variables).

Figure 2 illustrates data processing using streams with an example. In that figure nodes were configured to collect, transform and deliver data into a visualization application. In the figure a signal sample is collected periodically by each of three sensors, placed into a stream and routed into a relaying node. The relaying node computes some statistics (e.g. avg or max) over the three values and forwards the resulting value into the control station. The control station keeps a stream with the last values that were received, and visualizes those values. It would be possible to configure differently any of the nodes in this illustrative example. For instance, sensor nodes could each compute some statistics over 10 samples and forward those to the control station directly, which would show the stream values or compute some other statistics.

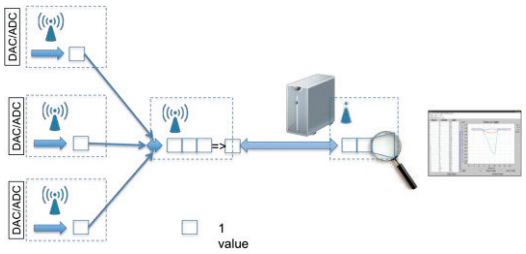

Figure 2 - Illustration of the Stream Processing Model

V. EXPERIMENTAL EVALUATION

In this experimental section we use a prototype to show that we are able to configure both WSN nodes and a control station using the same interface with simple and uniform API calls. We also show that the configuration resulted in corresponding behaviour modification.

We have implemented a system configuration and operation prototype as a proposal within the European FP7 research project Ginseng on performance-controlled WSNs, in which applicability in actual industrial scenarios is an important aspect. In that setting, WSN-LightProcessor allows users to configure and change what any part of the system is expected to do easily and remotely, concerning data collection, alarms and actuation.

We have used a lab testbed to test our approach. The testbed is a totally planned network (Ginseng focuses on totally planned networks with performance guarantees) with 16 TelosB nodes organized hierarchically in a 3-2-1 tree and a control station receiving the sensor samples and alarm messages. The WSN nodes run the Contiki operating system with a TDMA network protocol (GinMac) to provide precise schedule-based communication. The TDMA schedule had an epoch time of 1 second. Nodes are time-synchronized and awake for their predefined slot time.

Experiments involved configuring nodes with the following alternatives:

1. A temperature reading is obtained every 3 seconds from every sensor node;
2. An alarm is raised in the control station when a threshold is passed.
3. An alarm is raised in a sensor node for a specified threshold;

We have written simple control station application that allows submitting commands and collected data logs in the control station to analyze the modifications to sampled data coming from the WSN. With this data we have built timeline charts to show the results.

As sensor nodes are programmable, we can configure those to send the temperature values periodically (every 3s), using the following method call:

```
Node.createStream(Zone1SensorNodes,
        "temperatureStreamFromZone1Sensors",
        3s,
        {(TEMPERATURE, VALUE)}
);
```

After the call to createMessage, the system configures the WSN node through a specific command message and the WSN nodes start to send data readings with the specified rate.

Figure 3 illustrates a timing diagram showing the instant of submission a command, the instant of reception then by destination node and the period of created message ("temperatureStreamfromZone1Sensors") for a node at third level of the tree.

The delay to receive the command at the destination node depends of the instant of submission and can vary between a few mili-seconds and 800ms. Figure 3 shows the average delay obtained for a test where 20 commands were sent.

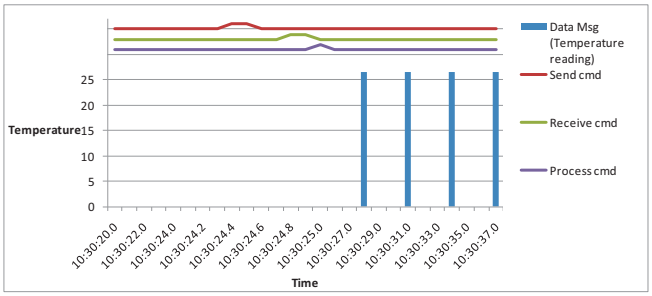

Figure 3 - Timing diagram

WSN-LightProcessor also provides functionalities to raise alarms if sensor data is above a certain threshold. These alarms can be raised on data source nodes (sensors), on any node in the path to the sink node or/and in the control station with different thresholds. The fact that whole parts of the system contain the same configuration and processing component and are directly referenced by an address variable allows uniform configuration in spite of being very different platforms (a TelosB node and a linux control station).

We can configure the control station to generate an alarm if the temperature value is greater than a certain threshold (for instance, 27 degrees Celsius) using the following method calls:

```
Node.createAlarm(controlSation,
        "ServerPressureAlarm",
        pressureStreamfromZone1SensorNode,
        CONDITION( TEMPERATURE.VALUE, >, 27)
        );
```

Figure 4 shows a timing diagram of event detection and alarm generation at control station.

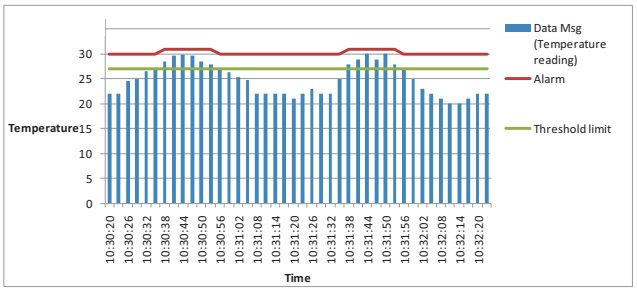

Figure 4 – Timing diagram for event detection and alarm generation at control station

The same configuration can be sent to the WSN nodes which allows to generate and send an alarm if the temperature value is greater than 27 degree Celsius.

Figure 5 shows the results concerned alarm detection time, measured as the time taken from an abnormal event happening at a sensor node and deliver the alarm notification at control station.

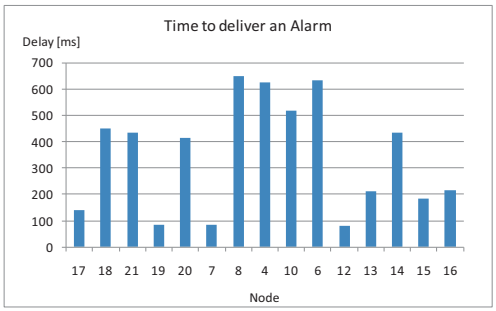

Figure 5 - Per-node alarm delivery time

The reliability is another import issue that refers to the amount of alarms that is sent by the node and successfully received by the control station. Figure 6 shows the reliability of the deployment concerning event detection and deliver notification.

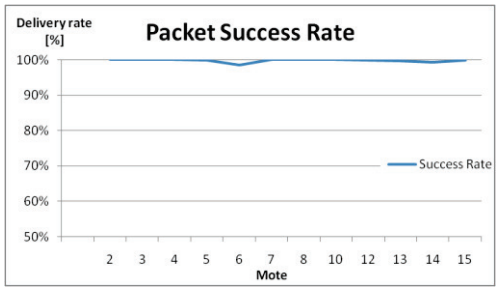

Figure 6 - Reliability

From the figure 6 it is possible to see that all nodes except node 6 have about 0.1% of losses. Node 6 has high losses, about 0.5%, but we assume that these losses are due external factors, such as noise due to the industrial environment.

VI. CONCLUSION

In this paper we proposed a middleware model for uniform configuration and operation over heterogeneous networks comprising WSNs and nodes outside the WSNs (control stations). The model advances the state-of-the-art since it views the whole system as a distributed system and any computing device as a node (inside or outside of the WSN, regardless of hardware or operating system) with the same configuration API, remote configuration capabilities and operation interface. This results in a lego-like system with great configuration flexibility and uniform API. We have described the modules and details of the component. Then we used our experimental testbed and defined a set of tests that show the system is able to configure both sensor nodes and control stations very easily and using exactly the same calls. From our test runs we extracted logs and displayed a timeline that proves correct configuration of both sensor nodes and control station using the approach. We also show results on delay statistics for alarm notifications.

REFERENCES

[1] C. Prehofer, J. van Gurp, and C. di Flora. Towards the web as a platform for ubiquitous applications in smart spaces. In RSPSI, at Ubicomp, 2007.

[2] V. Trifa, S. Wieland, and D. Guinard. Design and implementation of a gateway for web-based interaction and management of embedded devices. In Submitted to DCOSS, 2009.

[3] Jari K. Juntunen, Mauri Kuorilehto, Mikko Kohvakka, Ville A. Kaseva, Marko Hännikäinen, Timo D. Hämäläinen, WSN API: Application Programming Interface for Wireless Sensor Networks. The 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'06).

[4] Å Östmark, J Eliasson, P Lindgren, A van Halteren, An Infrastructure for Service Oriented Sensor Networks, Journal of Computers, 2006.

[5] Marco Sgroi, Adam Wolisz, Alberto Sangiovanni-Vincentelli and Jan M. Rabaey, A Service-Based Universal Application Interface for Ad-hoc Wireless Sensor NetworksIn Ambient Intelligence, W. Weber , J.M. Rabaey, E. Aarts (Editors), Springer 2005.

[6] Sam Madden, Michael J. Franklin, Joseph M. Hellerstein and Wei Hong. TinyDB: An Acqusitional Query Processing System for Sensor Networks. ACM TODS, 2005.

[7] JSR 256: Mobile Sensor API. JSRs: Java Specification. jcp.org/en/jsr/detail?id=256, 2006.

[8] S Krishnamurthy, TinySIP: Providing Seamless Access to Sensor-based Services, Mobile and Ubiquitous Systems: Networking & Services, 2006.

[9] A. Dunkels, J. Alonso, T. Voigt, H. Ritter, J. Schiller, "Connecting Wireless Sensornets with TCP/IP Networks", in Proceedings of the Second International Conference on Wired/Wireless Internet Communications (WWIC2004), Frankfurt(Oder), Germany, February 2004.

[10] L. Shu, J. Cho, S. Lee, M. Hauswirth, L. Zhang, "VIP Bridge: Leading Ubiquitous Sensor Networks to the Next Generation", Journal of Internet Technology, vol.8, 2007.

[11] K. Aberer, M. Hauswirth, A. Salehi, "Infrastructure for Data Processing in Large-Scale Interconnected Sensor Networks", in Proceedings of the 8th International Conference on Mobile Data Management, Mannheim, Germany, May 7-11, 2007.