Abstract—This project presents an easy and quick solution for the development and implementation of Wireless Sensor Networks (WSN). Wireless sensor networks are used to monitor the environment around us. Normally used to prevent disasters in remote areas such as forests, mountains or deserts. With the recent technological advancements, sensor networks found a variety of applications use cases in everyday places like hospitals, factories, buildings, and in recent years inside our homes. Despite such diversity of applications, a WSN project is somehow arduous, and the tendency is for them to grow in numbers and complexity.

Usually a WSN project takes three steps, the designing process, the implementation process and its deployment, where most of the energy and time is consumed in the implementation phase, needing experienced users in programming and in WSNs, normally resulting in rigid solutions, where the possibility to modify the WSN behaviour is limited, or non-existent. This arises an obstacle for an already deployed WSN when the running application is no longer needed, or in the event the developed application’s objective has changed, requiring to go back to the designing process.

What this project proposes, is a system that offers a rapid and easy applications development environment for WSNs, where novice or experienced users in a simple and clean web interface using drag-and-drop visual programming model, are able to create and develop applications in real-time for WSNs.

Index Terms—WSN, RWSN, Quick Development, Interface, TinyOS, Modular, Visual Programming

I. INTRODUCTION

Wireless Sensor Networks (WSN) are used in a variety of applications, nowadays with technological advancements, this kind of networks can be found in everyday places, in which most of the time we do not even realize where they are, or their use. In the past, WSNs were used to monitor remote locations with difficult access, such as forests, deserts or mountains, collecting data such as humidity or temperature, light intensity, or quantity of carbon monoxide, where users took decisions based on the collected data.

In recent decades the technological advancement has allowed having reduced-sized microcontrollers with higher processing capabilities, faster and at lower costs. This improvement, as in other technological areas, has enabled the evolution of WSNs. Now with more capabilities, these networks are able to withstand a larger number of sensors, make more calculations, some even able to have some kind of artificial intelligence, enabling to diversify the ways of how WSN are used.

This type of networks can go from collecting data of a single sensor to an extensive and complex management factory control network, being found in manufacturing lines, enabling the automation of repetitive work, or replacing workers to the exposure in dangerous situations; Warehouses for maintenance and replenishment of stocks; In buildings to improve the energy efficiency, or controlling the access of personnel; In hospitals for monitoring patients; Or more recently in household environments to improve our daily lives. These were, not to mention more, some of the examples where the WSN are now being used. The proliferation of WSNs, enabled their use in more applications, resulting in larger and more complex networks.

Usually in a WSN project, the first phase is to define the WSN goal, the second step is to choose the hardware and software tools to start developing, next is the designing and programming of the WSN nodes, following with its implementation and deployment, which normally results in a rigid solution, and afterwards when a modification to change the WSN behaviour is needed, it is necessary to go back to the designing and programming phase. With this approach, as the complexity of the WSN grows, the cost of developing software increases dramatically.

To overcome these issues, this paper presents SADUM, a visual development framework capable of speeding up the process of application development for WSNs. Its an open-source, platform-independent, extensible and scalable development framework, that allows users, with or without experience, to quickly create WSN applications in real-time, using a visual tool. This visual tool uses a visual block programming model as an abstraction of the capabilities of the nodes, in which users by connecting them form diagrams, a description of the program logic, that allows the application development process to be faster and easier, instead of the traditional text programming approach. This way users can create applications for WSNs in an easy and more intuitive way, without the need to write a single line of code, only focusing on the problem, instead of being worried about how the WSNs works.

II. RELATED WORK

In this section are presented some tools that use some kind of visualisation approach to developing applications for WSNs.

TOSDev [1] - It is an IDE (Integrated Development Environment) of NesC programming language for TinyOS operating system, that allows TinyOS developers to reduce application development time, by creating an abstraction layer over the NesC. This abstraction virtualizes NesC components, by creating a graphical representation using blocks. This allows users...
to create applications in a visual manner, all integrated into an IDE, that enables graphical editing and wiring diagrams of the blocks.

Viptos [2] - It is a JAVA IDE, and like TOSDev uses a block diagram programming approach to generate source code for WSN nodes running TinyOS. The main feature of this tool is the ability to simulate sensor networks and sensor data, allowing the simulation of heterogeneous networks, providing an interrupt-level simulation of actual TinyOS programs, with packet-level network simulation.

TinyInventor [3] - Is an open-source cross-platform development environment, based on Open Blocks, that provides drag-and-drop cascading blocks diagram programming language integrated into an IDE. This approach differs from the other tools because it allows users to quickly develop applications, by visualizing the application as a whole instead of multiple applications entities, in which the placement of blocks defines how the source code is generated and compiled. Another feature of this tool is the ability to monitor the WSN, displaying the collected data in plots.

MARWIS [4] - Is a management architecture for heterogeneous WSN, which allows tasks such as visualization, monitoring, reconfiguration, updating and reprogramming of a WSN. Its main feature is the hierarchical architecture using a wireless mesh network, that operates as a backbone, dividing heterogeneous WSN into sub-networks, each containing sensor nodes of the same type, where the users are able to manage such tasks using a GUI.

Octopus [5] - Is a monitoring, visualization and control tool for WSN implemented in NesC for TinyOS, its main objective is to provide flexible access and control of deployed WSN, by providing a GUI where users can control the behaviour of the network, getting live information about its topology, and sensor data. This GUI allows users to some extent, control and change some node behaviours like changing the sampling sensor periods, setting sensing thresholds, modify the sleeping duty cycles of the nodes, or change the radio frequency channel, having also the ability to plot the gathered sensors data.

IoTSys [6] [7] - It is an integration middleware for IoT (Internet of Things), providing an IPV6-based communication stack for embedded systems, Web services, and Obix, enabling interoperability between smart objects, focus on building automation technologies. It uses 6LoWPan and CoAP communication protocols along with EXI format, allowing an efficient communication between heterogeneous smart objects, enabling to operate with sensors and actuators through a graphical programming Web technology.

USER-FRIENDLY PROGRAMMING FRAMEWORK FOR WSN [8] - Is a framework based on PROVIZ [9] framework for devices running TinyOS. This framework integrates a network visualization tool enabling the comparison between two WSNs, analysing the link characteristics of the deployed networks. Also having a scripting language programming tool, and a visual programming tool to help reduce the time of WSN development.

The mentioned tools were developed to simplify the task of programming WSNs, all providing a graphical programming approach for the applications development process. However, none of this tools completely address all the existing difficulties at the time of the implementation and development for WSNs. The TOSDev and Viptos IDEs require an intermediate experienced user for applications development. TinyInventor being one of the easiest to use, grouping with the two previous tools, are not able to reconfigure a WSN in real time, and every time an application is modified it requires to be uploaded node by node, something undesirable in a WSN. The MARWIS tool disadvantage is in the necessary infrastructure for its implementation, and like Octopus, they were developed for monitoring purposes, lacking the reconfiguration ability, only allowing some WSN characteristics to be modified. The User-friendly tool has the same issue related to the WSN reconfiguration of the first three tools, also lacking in configuration options in the GUI. For last the IoTSys being the most integrated solution, the main downside is that the smart objects for its implementation must use the CoAP, KNX and OBIX protocols, which the last two are not commonly implemented for most of the devices used in WSNs.

What SADUM presents is a solution to make the development and deployment of a WSN easier, presenting a clean and simple Web-based GUI solution, providing a visual programming model using blocks, that allows the users to create applications for WSNs in real-time.

III. Architecture

SADUM is intended to create a full system to make easier the development of WSN, suited for users with any level of experience. The main objective is to aid in the initial impact on any WSN, its programming. This is accomplished using a GUI, enabling users to create applications in a more intuitive way, this interface utilizes technologies that enable its use in a larger number of devices, such as computers, tablets or smartphones, having the ability to be accessed remotely, where users can develop applications in real-time, not requiring physical access to the sensor nodes.

As shown in Figure 1, the architecture contains the following elements: a WSN, a Sink Node, a Gateway, a Server and a GUI.

A. WSN

The WSN is composed of sensor nodes, in which SADUM implemented using the two following platforms, Micaz [10] and Iris [11] Motes, both running TinyOS [12] operating system. These motes encapsulate all the functionalities of a sensing node, being able to read sensors, control actuators, and communicate with other nodes. The software running on the motes was developed using a modular architecture, where each module represents a node capability, where a capability provides a node functionality like reading a sensor, acting on an actuator, or sending data to another node. This type of architecture allows the node to be reconfigurable at runtime, where at user request these capabilities are advertised to the terminal, where users using a visual abstraction of
those capabilities in the GUI are able to easily reconfigure the applications running on the nodes.

B. Sink Node

This element was developed to act as a medium of communication between the WSN and the Gateway, just passing the messages from the Gateway to the WSN and backwards. The Sink node is connected to the Gateway through serial communication (i.e USB) and communicates with WSN via Radio communication. This node is not reconfigurable and has no sensors or actuators.

C. Gateway

The Gateway is typically located in a specialized node or in a computer, in SADUM it was implemented using a single board computer, with serial and internet communications, acting as a bridge between a WSN and Terminals (Users), via Server. At startup this element recognizes the WSN, discovering the network nodes capabilities saving it in memory, and when requested by the user delivers it to the Terminal GUI. The Gateway is also responsible for handling the communications with the Server, as well to convert the applications created in the GUI by the users, to a message byte format to reconfigure the network nodes. Another function of this element is to save the GUI state, like the nodes applications and the collected data from the sensors.

D. Server

The Server main functionality is to act as a central device where Users and Gateways (i.e WSN) connect to, providing a single point of interaction between this elements. Another function of the Server is to provide the web-based GUI to the Users, where using a simple interface Users can consult how many WSNs can have access to. This architecture however normal is not usually used in WSNs, where most of the solutions apply one Server per Gateway per WSN, so in SADUM to allow multiple users get access to multiple Gateways (WSNs) using a single address, a Sever centric architecture was implemented.

E. Terminal - GUI

The GUI is embedded in a Web page stored in the Server, this page is where all WSNs are displayed, allowing Users to choose which of the available WSNs want to access.

This GUI also provides a visual programming tool, where users can develop applications for the WSNs. This tool using visual block elements as a representation of the capabilities of the nodes, allows users to describe the program logic in an intuitive way by using block diagrams.

IV. IMPLEMENTATION

As previously mentioned SADUM is composed of 5 elements, in this section it is explained how these elements work, and how the reconfiguration process is done.

A. WSN

A WSN usually is composed of multiple nodes, with one acting as a Sink Node and the remaining ones acting as a sensory/actuator type of nodes. These elements were all implemented using Iris Motes, running TinyOS operating system, a widely used OS for resource-constrained hardware such as WSNs. Each of the nodes, except for the Sink Node, had the MTS300[13] sensor board mounted, composed of one temperature sensor, one light sensor and one buzzer.

1) Nodes: As SADUM serves to help users in developing applications for WSNs, and this was achieved by developing a software capable of being reconfigured using messages. The software running on the nodes uses a modular architecture, where the root of the program is in charge of managing various modules. A module, see Figure 2, can be viewed as an isolated block of code with some logic inside, that can read a sensor, make a calculation or send and receive messages. Blocks communicate with each other through an interface, called SadumConnectionEvents, that can be thought as ports, channels of communication between blocks. This interface is composed of two kinds of ports, Inputs and Outputs, where each port can send or receive three types of data, an 8-bit integer, a 16-bit integer and a boolean type. Each block depending on its functionality must implement its
own SadumConnectionEvents logic, setting the ports to get or generate events. Figure 2 shows three examples of blocks, at the left an Input block type that generates output events, this type of blocks were created to begin the program flow normally used to read sensors. At the right is the Output block type, that receives events from other capabilities, this type is used to operate as actuators. In the centre is the Function block, having both Input and Output ports, it receives events at the input ports, perform some kind of operation on the received data and forwards the results at the Output ports.

These blocks represent node capabilities each having its own functionalities, where one node can be composed of multiple capabilities that can be repeated. For this work, 24 node capabilities were developed as described in the following list:

- **TIMER_MILLI** - This capability implements a Timer, firing a Boolean TRUE whenever the timer expires.
- **LEDS** - Implements a controller for the 3 LEDs presented in the Motes boards, by setting the least 3 significant bits of an 8-bit integer, turning ON and OFF the corresponding led. To set a led ON the corresponding bit must be set to 1, and 0 to OFF.
- **LED_RED, LED_GREEN and LED_YELLOW** - These three capabilities control each led individually with Boolean values. TRUE to set ON and FALSE to set OFF.
- **PHOTO** - This capability is for reading the light sensor, that can be found on the MTS300 board, reading values every 250 ms.
- **TEMP** - This reads the temperature sensor, also found on the MTS300 board, having the same reading rate as the PHOTO capability.
- **SADUM_SOUNDER** - The buzzer on the MTS300 board is controlled by this capability, where the 8-bit integer value received at its input port determines the duration of the buzzer sound.
- **RADIO_SEND and RADIO_RECEIVE** - These two capabilities are able to send and receive messages to and from other nodes, both allowing the user to choose from which node the messages should be sent to or receive from. These capabilities only work in pairs, a SEND must be paired with a RECEIVE capability for a message to be transmitted.
- **16_TO_8** - This capability converts a 16-bit integer to an 8-bits integer by shifting the 8 most significant bits to the right.
- **LOGIC_GATE_NOT, LOGIC_GATE_AND and LOGIC_GATE_OR** - These 3 capabilities implement the behaviour of the NOT, AND and OR logic gates with boolean values.
- **ARITHEMETIC_INT8 and ARITHEMETIC_INT16** - Implement the sum, subtraction, multiplication and division arithmetic functions, the first over 8-bit integers and the second over 16-bit integers.
- **RELATIONAL_INT8 and RELATIONAL_INT16** - These capabilities implement the greater (>), greater than or equal (≥), less (<), less than or equal (≤), equal (=) and not equal (≠) relational functions over 8-bit and 16-bit integers.
- **CONSTANT_INT8, CONSTANT_INT16 and CONSTANT_BOOL** - These capabilities are equivalent to 8-bit and 16-bit integers and boolean variables.
- **TRIGGER** - This capability consists of a three ports block, with 2 inputs and one output. Whenever the trigger input port receives a boolean TRUE value, it forwards the value of the second input port to the output port.
- **CHART** - This capability sends any value at its input port to the Gateway, to be saved and displayed in the GUI. Its function is to monitor the values collected by the WSN nodes.
- **MULTIPLEXER and DEMULTIPLEXER** - The first implements the function of a multiplexer electronic device, that selects one of the several input ports, forwarding the value to the single output port. The second implements the opposite function of the multiplexer.

**Reconfiguration Process**

The reconfiguration of the nodes is done using KLV (Key-Length-Value) messages, based on the RDL [14] work, this format describes node resources or services, such as sensors, values or functions, in a byte compact format that can be used in systems with constrained resources such as WSN.

The KLV format associates a variable length value with a fixed size integer key identifier while describing the size of the value as a triplet. An item specified with KLV is encoded into a Key-Value-Length triplet, where the Key identifies the type of data, Length the data length, and Value the data itself.

The node capabilities are encoded using a KLV tuple structure like the one shown in Table I. This structure is composed of 6 bytes having an outer KLV with an inner KLV tuple. The outer KLV identifies the message as a SADUM_NODE_COMPONENT type, the inner KLV identifies the node and one of its capabilities.

Each node and capability have unique IDs, the first is unique in the WSN, and the second is unique in the node, generated at bootup, this allows a capability to be unique inside the whole WSN. This structure, when requested, is inserted in a message and then sent to the Gateway, that is in charge of converting
this messages into visual elements that can be used by the GUI.

To make a node behave like the applications developed in the GUI, the capabilities have to know how, and to which capability should send and receive events, this is done using the SADUM_CONNECTION type of messages which have two structures, as shown in Table II. The top message structure is sent at the beginning and at the end of each reconfiguration process, in which the first byte notifies the node to process the received message as a SADUM_CONNECTION type, the second byte, when set to 0, tells that a configuration process is about to start or to end, depending on the value of the third byte, 0 to begin and 1 to end.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Level</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 1 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADUM_NODE_COMPONENT = 0</td>
<td>Key</td>
<td>Length</td>
<td></td>
</tr>
<tr>
<td>LENGTH = [0, 255] x Value (Nível 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADUM_NODE_COMPONENT ID = 0</td>
<td>Key</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>SADUM_NODE_COMPONENT TYPE = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In between the start and stop reconfiguration messages, a node can receive any number of messages with the bottom structure of the Table II. This second KLV message structure formed with 12 bytes is organized in 4 KLV tuplet levels, that describes how capabilities are connected.

The outer KLV identifies the message as a SADUM_CONNECTION type, making the node aware that a connection-related message was received, the next KLV level identifies the message to be of a connection creation type, when the Key value is set with 1. The third level is formed by two KLV tuplets, where the source and target ports of the connection are defined. Both source and target KLVs tuplets have the same information, the node capability ID and its port ID. With this, the node is able to form a connection between capabilities, where the port of a target capability is set to only receive events from the source capability port.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 1 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADUM_CONNECTION = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADUM_CONNECTION Create = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADUM CONNECTION Create, Source = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADUM CONNECTION Component = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component ID = [9, 0, 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADUM CONNECTION Port = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Port ID = [0, 0, 0, 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADUM CONNECTION Create, Target = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADUM CONNECTION Component = 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component ID = [9, 0, 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADUM CONNECTION Port = 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Port ID = [0, 0, 0, 0]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the end, a stop reconfiguration message is sent by the Gateway to inform that the reconfiguration process has ended, in which the node signals all capabilities to start generating events, that in this example the Temperature Sensor is the first one, generating an event in every 250 ms, forwarding the readings to the 16 To 8 bit Converter capability, that in turns forwards the converted data to the Chart capability, finally sending the read data back to the Gateway, where the collected data can be displayed in the GUI.

This example illustrated how a node of a WSN is reconfigured, using small sized messages, in KLV format, allowing a fast reconfiguration process in real time without the need of reprogramming the nodes with new program images.

2) Sink Node: The Sink Node is loaded with a program that functions as an intermediary of messages, having two buffers, one for messages received from Gateway to be sent to the WSN nodes, and a second buffer for messages received from the WSN nodes to be sent to the Gateway. Because of the number of messages passing through this element, two independent buffers were used, to prevent the messages loss from the Gateway, due to the node hardware, it takes a lot of processing time when reading from the serial communication channel. This node is connected to the Gateway through serial communication available from the MIB520 [15] and communicates with the WSN nodes via radio, this node has no sensors and is not reconfigurable.

B. Gateway

The Gateway is the element of this architecture that holds the information about the WSN, the number of nodes, their capabilities and any collected data. The Gateway is the route of communication between users and WSNs, being connected to the Sink Node and the Server, was implemented in a UDOO [16] Quad computer board, that allows having some computational power and at the same time to not overload the Server. The software developed for the Gateway used the Node.js [17] framework, that is a JavaScript development
environment, which runs JavaScript code as an application, allowing to reduce the programming languages used in this project.

This element is able to recognize the WSN nodes and their capabilities, translating them into JavaScript objects that are used by the GUI to create the necessary visual block elements, that users can use to develop applications. Another function of the Gateway is to manage the reconfiguration process of the WSN nodes, converting the applications created in the GUI into KVL messages, recognizable by the nodes.

All the information about the WSN and the GUI applications are managed and stored in the Gateway, as the collected data from the nodes. The Gateway is also able to save the GUI application development state, and when asked to retrieve to the user.

C. Server

The Server acts as a central point, connecting Users and Gateways. The main functionality of this element is to provide the GUI to the Users and manage the Users-Gateways access. This element was developed using the Node.js and Express.io [18] frameworks, the last framework was used to create a JavaScript web server allowing to answer HTTP requests.

Whenever a new user connects to the Server, it hands over the initial GUI web page and requesting at the same time to all connected Gateways to identify themselves to the new user. This initial page immediately allows the users to see how many WSNs are available. After this, if a user chooses to connect to a WSN, the server checks if is occupied by another user, if not the Server sends the second web page where the actual GUI for applications development resides, creating a new communication channel where this new User and a Gateway can exchange messages.

D. GUI

The GUI (Graphical User Interface) is where the Users are able to visualize and develop applications for the WSNs. The GUI was embedded in a normal Web page, developed with HTLM5, JavaScript, JQuery and CSS technologies, together with JointJS [19] and ChartJS [20] frameworks.

The GUI is composed of two Web pages, the first one shows all the WSNs connected to the Server and the second one is where the User can compose and manage the applications for the nodes.

Presentation Page

When users access the Server, they are handed with the page shown in Figure 4. In this page, users can immediately see how many WSNs are available.

Each WSN is represented by a small panel, whereby clicking on it, the users get a brief description of the WSN goal and the option to connect to it. If the user chooses to connect and if no other user is currently connected to that WSN he will be presented with the second Web Page, the Network Page.

Network Page

In the Network Page, after the login, the user is presented with this second Web Page, in which is able to visualize all nodes composing the WSN, develop applications for the WSN nodes and monitor all the gathered data, where each of these features is accessible in three distinct views, the Network View, Node View and the Monitor View.

Network View

This is the first view shown to the user, see Figure 5, where he is able to modify some characteristics related to the WSN. This view is composed of a top bar where the user is able to perform some actions, and a white area, that shows all the WSN nodes.

In the top bar, the user has access to actions like a tutorial menu explaining how the GUI works, a menu to change the WSN description that is shown on the Presentation Page (Figure 4) as well to set a password to access the WSN. An action to refresh the WSN nodes, whenever the user wants to add or remove a node from the network, an upload button to reconfigure all WSN nodes at once, a sign-out button and an action to upload a background image.
In the white area are shown all the nodes of the WSN, represented by green circles, where the user by repositioning them can create a visual representation of the WSN topology, like the example shown in Figure 8.

Node View

The Node View, as shown in Figure 6, is where the applications development occurs, having each node its own Node View. This view is composed of a top actions bar and an application development area.

In the top bar of this view, the user has access to a side menu, where all node capabilities are shown, like the right side menu in Figure 6. These capabilities represented by blocks are the visual representation of the developed capabilities mentioned in the Sub-subsection IV-A1, being divided into three categories, Inputs, Outputs and Functions. Each capability, if clicked, allows the user to see a brief description of its functionality in a small panel that appears at the bottom left side of the view, as shown in Figure 6.

To start developing applications the user just has to drag the blocks from the side menu and drop them in the application development area, the blocks in this area are the ones that will be active when uploaded to the nodes.

The blocks have input and/or output ports, each filled with a colour, representing a data type that can receive or forward, where a red port is associated with a boolean data type, a dark blue port to an 8-bit integer, a light blue port to a 16-bit integer, and a special port filled in white, that can receive or forwards all data types.

The connection between blocks have rules as follows: Input ports can only connect to one Output port; Output ports can be connected to multiple Input ports (fan-out); Ports cannot connect to ports of the same block; and finally, Ports can only connect to ports of the same colour (i.e data type), except for the white ports.

When developing the applications the GUI will help the user to follow this rules with visual hints.

The diagrams formed by the blocks connections, allows the user to easily understands the logic and the different flows of the application. At the end, the developed applications should be uploaded to the nodes.

Monitor View

The Monitor View is where the user has access to visualize the data collected by the WSN nodes in a Chart form, see Figure 7. This view is only available if one of the nodes applications has the Chart block in the application development area, like the rightmost block in Figure 6. The function of this block is to forward any received data at its Input port back to the Gateway, which forwards the collected data to the GUI.

This view allows any number of charts, from the three available, a Line Chart, a Digital display and a Polar Chart, which enables the user to choose the more suitable way to display the collected data.

V. User Experience Evaluation

In this section is described the experiment conducted with real users in order to validate the SADUM system, the objective of this experiment was to evaluate the degree to which SADUM can help users in developing applications for a real WSN.

For this experiment, a WSN was installed in a room, composed by 3 Iris motes, each with the MTS300 sensor board mounted, all connected to a constant power source and loaded with the same number of capabilities. It was conducted with the help of 20 experienced and novice users random selected.

The experiment consisted of asking users to develop 4 applications for the installed WSN, following the instructions from pictures. In each application, the difficulty and complexity of the programs grew, allowing users to perceive how the GUI worked and how the applications development was made.

The first application consisted in turning a led ON and OFF, in which the objective was to allow users familiarize with the menus and the applications development process.

The second application was to demonstrate how to collect data from a light and temperature sensor, displaying it in a chart.

The third application was to replicate a real use of a WSN, by creating an application to measure and calculate the mean value of the room temperature and displaying the collected data.
data in a chart, using all the WSN nodes. This application allowed users to understand how the communication between nodes worked.

The previous exercise’s objective was to help users understand how to develop applications using the SADUM GUI, which terminated with the last exercise where users were asked to develop an alarm application without the help of a picture. This fourth exercise, allowed to evaluate if the capabilities developed for the node were suited for the user’s needs, where it was interesting to see different solutions created by the users for the same problem, even though users had access to the same node capabilities. This exercise also proved the robustness of the system, where it was not necessary human intervention over the course of tests.

At the end of the tests, participants were asked to answer a questionnaire based on the SUS (System Usability Scale). The average SUS score of 68, according to the SUS scale metrics is considered an Average system, so with the score of 80 for this work, the usability of this system is considered to be Good by the users, which results, after the conversion to a percentile rank through a normalization process in a 90%, resulting in a grade of B.

VI. USE CASE

In this section is shown an example of a household scenario using SADUM system where a user wants to implement a WSN in his living room, turning it in a more enjoyable and intelligent room. It is important to remark that this use case is not to present the best solution for the application purposed, but to show the potential of the developed system in a plausible scenario of use.

As in all systems, there is a first phase, the installation of the system, where the user has to decide how many devices to use and in which to install each of the SADUM elements. The SADUM system allows the Server or the Gateway to be installed on the same device or in independent devices, leaving the choice to the user, regardless of the installation mode, the system architecture is always maintained.

After this process the user has to install the program that will run on the nodes, loading the Sink Node with the respective program, and on the remaining nodes the user, as he is not able to predict what application to develop, he loads the nodes with 40 capabilities, of the 24 developed. At this point the user is able to deploy the WSN, spreading the nodes through his living room, as shown in the Figure 8.

At this stage there are no applications running in the WSN nodes, so the user decides to start developing an application where he can monitor the room temperature, and by trial and error he achieves his objective, but as the time goes by, he realizes that this application does not help him be more fond of the room.

Then by checking the room, the user realizes that one of the nodes is near the windows, so he begins to question if that node can be used to control the windows blinds, so he develops an application to maintain the room temperature by controlling the windows blinds height with the light captured by the light sensor, obtaining the application shown in Figure 9.

In this figure are described the applications running on Node 1. On the left group of blocks is the first out of three parts of the temperature monitoring application, this group is just collecting the data from the temperature sensor, sending it to Node 2. The right group illustrates the windows blinds control application, which compares the values of the light sensor with a predefined value set by the user. If the sensor value is higher than the predefined value the blinds would get lower, decreasing the room temperature, if the value is lower the blinds would raise, increasing the room temperature. To note, in this example the actuator for the windows blinds is represented by a led, but this could be connected to a real switch.

After this, the user starts to see some improvements when using the room, but still unhappy, sees a way to improve the efficiency in maintaining the room temperature, and starts to develop an application for the node near the air conditioning system.

Realizing the value of the windows blinds and the air conditioning applications in his the living room, now what concerns the user is the security of the room when he is out for work. So the user checks the security systems available on the market and realizing that they are too expensive, he thinks of a way in using the already installed WSN in his living room and develops a simple alarm application using 2 nodes buzzer. Even though it does not prevent people from entering it can signal the user that someone has been inside. Obtaining the applications for his WSN as shown in Figures 9, 10 and 11.

In Figure 10 are the application running on Node 2, at the top left group of blocks is the second part of the temperature monitoring application, which reads the temperature sensor,
adding it to the value received from Node 1, forwarding the result to Node 3. In the same figure, the bottom group of blocks is the first part of the Security System application, which fires the alarm whenever the value read by the light sensor is higher than a predefined value set by the user. This group also has a switch to activate the security application.

In Figure 11 are the applications running on Node 3. On the top group of blocks is the last part of the temperature monitoring application together with the Air conditioning application. This group receives the value from Node 2 adding it to the temperature sensor value read by this node, which calculates the mean room temperature value by dividing the sum of all temperature sensors values by three, forwarding the result to the Gateway that will display the mean value in the GUI. The mean value is also used to feed the Air conditioning application, which is similar to the windows blinds application, but this time the user sets the activation of the Air conditioning system when the mean value is higher than 24°C.

At the bottom group is the second part of the Security System application, that is the same as the one running on Node 2, the reason to run the same application on two nodes is to improve the accuracy of the Security System application by creating two trigger points.

This Use Case helped to validate the SADUM system in a household environment example, where it was shown that a WSN can be programmed without the need of writing a single line of code, in which a user simply by using a visual programming model can rapidly develop complex applications for an entire WSN.

This scenario also confirmed that SADUM is able to adapt to various problems, attaining to the user needs as they change, being able to be used by all types of users.

VII. CONCLUSIONS

SADUM is a project that tries to help users, to develop and implement a WSN, using a simple graphical user interface and a visual programming model based on blocks.

The motivation behind this project was to create a solution capable of being fully reconfigurable and adaptable to any objective, for the resource-constrained devices used in WSNs. SADUM is an open-source cross-platform presenting a full system composed of a WSN, a Gateway, a Server and a GUI. In which the nodes developed with a modular architecture are able to advertise their capabilities to a simple and clean GUI, where users using a visual programming block model, are able to easily create and develop complex applications for an entire WSN, in real time.

This project was evaluated by taking two approaches: developing applications in a household scenario, to exemplify its use and potential, and by conducting an experiment with real users to perceive the GUI user experience.

The objective of the use case was to explain the adaptability of the system, by giving a real example in a household scenario, where the WSN has to be able to adapt as the user needs changes, demonstrating that by using a visual programming model approach, a user can easily modify the WSN, being able to easily develop several applications for a room in a fast and intuitive manner, being able to create complex programs without the need of experience either in WSNs or coding.

The second experiment with a population of 20 users, both beginner or experts in WSN field, consisted of asking them to develop 4 applications using the GUI. The purpose of this experiment was to measure the GUI usability, to see in which way this programming model can help users in developing applications. The results indicated the satisfaction of the users with this programming approach, even when they were not familiar with WSNs, stating that they could easily use SADUM for WSN applications development.

At this moment SADUM can be found in an IST-TagusPark room, with a 3 nodes sensor network, running a temperature monitoring application.

Although this project succeeded in achieving all the goals to which it was proposed, presenting a complete solution able to suppress some of the difficulties when developing a WSN, there is still room for improvement, namely to implement a communication protocol for the nodes to ensure the message delivery and develop more node capabilities. An interesting use for SADUM could be to implement the WSN nodes using other types of hardware, such as Arduino or the Wifi module ESP8266, using the proposed software modular architecture.

Finally, it would be interesting to implement this solution in real locations, such as factories, public buildings, or even in remote locations to evaluate the system behaviour in large-scale scenarios.
REFERENCES


