CoMaDa: An Adaptive Framework with Graphical Support for Configuration, Management, and Data Handling Tasks for Wireless Sensor Networks

Corinna Schmitt*, André Freitag† and Georg Carle†
*Department of Informatics, Communication Systems Group,
University of Zurich, Switzerland. Email: schmitt@ifi.uzh.ch
†Department of Computer Science, Chair for Network Architectures and Services,
Technische Universität München, Germany. Email: andre.freitag@tum.de , carle@net.in.tum.de

***Abstract*—Nowadays, users request comfortable frameworks and development environments independent of the applications. Those solutions should offer as many support as possible, should be user friendly, and allow manipulation and visualization of different things at the same time. Those requirements become very important in the area of wireless sensor networks due to different vendors, wide range of application field, and the high amount of collected data. Currently existing solutions cover only a limited range of service and are usually fixed on hardware, operating system or application. In order to offer the user the wide range of flexibility the CoMaDa framework was developed, which is presented in this paper and combines all above mentioned user requirements by working with virtual representation of real wireless sensor networks and support in real time. CoMaDa offers the user support for configuration of components, network management functionalities, and data visualization at the same time. CoMaDa is build in a flexible way, which allows the user to integrate new features (e.g. personal code, hardware support, visualization option) with less input and, therefore, adapt the existing CoMaDa to every setup as requested.**

I. Introduction

Today the call for comfortable frameworks and development environments occurs independent of the application nearly everywhere. Users want to work with intuitive tools, which cover as many as possible aspects of the task they are facing (e.g. offering programmatically interfaces, visualization tools, debugging tools, etc.). These requirements especially apply to wireless sensor networks, where a variety of different hardware, operating systems and protocols complicate the development of software. Many individual solutions, specializing in specific hardware/platforms or designed for specific tasks (e.g. simulation, visualization), exist and support the user in developing applications for specific platforms. Since the user should not be forced to switch to a different environment whenever he decides to use different hardware/platforms, the authors decided to develop a unifying framework that allows to easily adapt applications to new environments, including different hardware, node platforms and protocols, allowing the user to develop applications regardless of the underlying network architecture.

The developed framework is named CoMaDa, which is an abbreviation for **Co**nfiguration, **Ma**nagement and **Da**ta handling Framework. CoMaDa offers a centralized way to design applications that interact with wireless sensor network including functionalities for data export and network control. Under the term *centralized* the authors understand in this case that the application runs on unconstrained devices outside the wireless sensor network. Due to this centralized construction it is allowed to apply logic to a wireless sensor network that needs an overview over the whole wireless sensor network. This overview can only hardly be done in a decentralized way. For example, in a centralized solution it is possible to power down specific nodes in regions that have enough redundancy, which also allows more intelligent observation schedules at the same time. Due to this fact CoMaDa allows to pull logic out of the wireless sensor networks resulting in leaving the sensor nodes more power for the important tasks (e.g. calculations, measurements). Additionally CoMaDa provides an adaptive graphical user interface, which provides unified management tools and can be adapted with few input for new platforms, operating systems, and algorithms.

The remainder of this paper is organized as follows: Section II will give a brief overview of different graphical user interfaces, which influenced the design decision of the developed CoMaDa in this paper. The design decisions and the finally realized architecture of CoMaDa are given in Section III. In order to give a proof of operability a building scenario was assumed and the manifold possibilities supported by CoMaDa is presented in Section IV. The presented work will be concluded in Section V. [1]

II. Related Work

Today different graphical frameworks for wireless sensor networks exist. Parbat et al. gave an overview of 18 visualization tools [2], which were analyzed. Those tools have in common that they focus mainly on simulation, routing, and visualization of sensor data. But they all do not offer a possibility to combine hardware configuration, network management, and data visualization at the same time. Users request these features today in

[1]This work was mostly done when Corinna Schmitt was with Technische Universität München [1].

order to administrate a system in a comfortable way and to be aware of system errors as soon as possible. At the same time the environment should be very flexible and independent in order to support different algorithms and hardware or operating systems to avoid using different instances of the interface in parallel. In the following a very briefly glimpse over existing visualization and simulation solutions is given.

The company Crossbow Technologies Inc. offers the interface MoteView for wireless sensor networks, which allows the user to integrate a client interface between the user and the network. MoteView offers tools that support the user by deploying and monitoring the networks. Additionally, MoteView allows the user to become connected to databases, to analyze the data, and to plot the data. MoteView was built in a modular way with four layers, which allows four visualization possibilities: Data, Commands, Topology, and Charts. The constraint for MoteView is the usage of sensor platforms of Crossbow (e.g. Mica, IRIS) and their supported operating system TinyOS. [3]

Due to user requests different simulators and emulators were developed in the community, such as NS-2, TOSSIM, OMNeT++, and COOJA. Those have in common that the user can setup a wireless sensor network with the personal code in order to evaluate its performance. In cooperation with the performance tests the user is able to vary a number of factors within the network (e.g. network size, deployment, delay, radio range, routing). All those simulators have in common that they are limited to one supported operating system. For example, COOJA supports only Contiki and TOSSIM only TinyOS. Sundani et al. pointed out in their research that no solution is applicable to all situations, which users can assume for the application. [4]

In order to sum up the related work it can be pointed out that no general solution exists, which allows the user to configure, manage, and to visualize data in any way at the same time. Additionally, no solution exists, which allows the previously mentioned requests to be studied independently of the operating system. Therefore, CoMaDa was developed and is characterized in detail in the upcoming Section III.

III. Architecture

First, this section presents the design decisions for CoMaDa based on the previously presented related work. Second, this section deeply characterizes the architecture of CoMaDa.

A. Design Decisions

Before dealing directly with design decisions for the implementation of a multi-purpose management framework for wireless sensor networks, such as CoMaDa, the requirements from users and vendors must be specified. In general, the following requests exist [1], [5]:

- Managing the wireless sensor networks including setup and manipulation tasks to network components even during runtime.
- Support of different hardware.
- Data visualization in real time (e.g. network status, collected data).
- Data export and import functionality in order to support visualization, analysis or feedback.

In order to support management functionality a programmable as well as a graphical interface is required. The programmable interface allows the user to implement and run specific program logic depending on the state of the network, whereas the graphical interface allows the user to react to visual feedback at runtime and also gives the possibility to easily setup the network.

Depending on the application scenario a wireless sensor network can or must exist of hardware from different vendors (e.g. IRIS and TelosB nodes from Crossbow Inc. [6], TelosB nodes from Advanticsys [7]). One reason can be the proof of program support on different platforms. Another reason can be a different sensor equipment of sensor nodes.

Additionally, also depending on the application, each network and each of its nodes fulfill a different purpose and, therefore, run with different software. One network may only be used to collect data about its environment (e.g. observing room temperature and humidity), whereas another network may be used to control an intelligent house. In a third scenario, the network may deal with critical data requiring additional security in the transmissions layer. On the one hand, in all those three cases the access to the data of the network and way of interacting with it highly differs, strongly depending on the used software. On the other hand, the requirements of the user to the network stay the same.

As mentioned in the beginning of this section, users generally want to visualize the network and the collected data, they want a way to configure the network components and to interact with the network. A wide range of such tools, like the ones discussed in Section II, exist but most of them are designed for a specific purpose or a specific architecture/software, forcing the user to either stick and specialize to one specific type of architecture/software or to switch tools from application to application.

The CoMaDa framework presented in this paper aims to offer an abstract interface to sensor networks that can be shared between different types of networks, allowing a user to write applications that dynamically adapt to different networks and therefore, offering the possibility to design applications that interact with sensor networks independent of the underlying network hardware/software. In order to achieve this, the framework adds an additional layer of abstraction, in form of a virtual representation of the network, between the part of the application, which defines the desired logic, and the part that actually interacts with the network. The second part can be seen as driver to the actual network, which strongly depends on the software and the type of the network but does not contribute to the logic and appearance of the application itself. Thus, it may simply be replaced when a different network is used, so the appearance and functionality of the application stays the same,

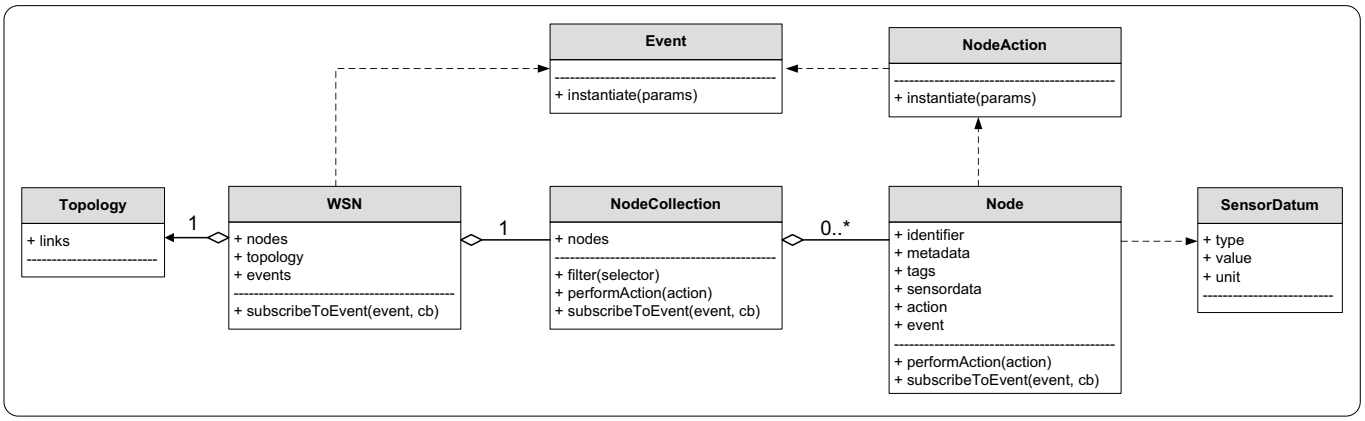

Fig. 1: Overview of the virtual sensor network representation used for CoMaDa

even so it is used with a totally different network.

The detailed architecture of the CoMaDa framework, which is written in Java, is described in the following Section III-B. In addition to the programmable interface of the framework, the previously mentioned graphical interface, designed as HTML/JS web-application, is presented as proof of concept of the framework in Section IV.

B. Architecture Characterization

In order to develop a virtual representation of a wireless sensor network, which can be used by applications to separate the program logic from the driver (actually interacting with the network), the common properties of all wireless sensor networks need to be specified. In general, a wireless sensor network is understood as a set of constrained sensor nodes, which are capable of observing their environment (e.g. measuring the room temperature) and communicating with each other, as illustrated in Figure 2 based on references [8] and [9]. In modern days, sensor nodes are not only restricted to observing, but they are also capable of interacting with their environment (e.g. controlling a light switch).

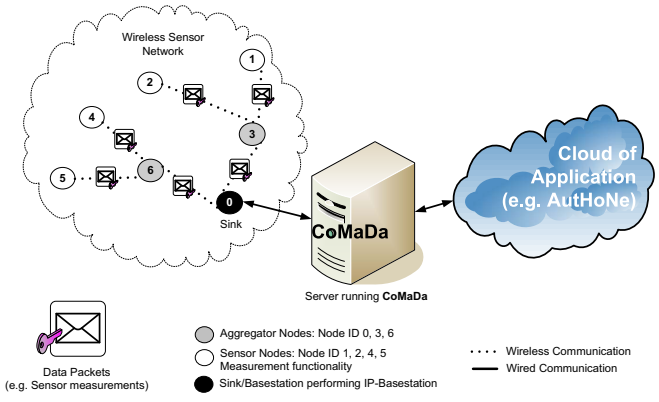

Fig. 2: Abstract overview of communication between sensors, CoMaDa, and cloud applications

In summary, a wireless sensor network can be defined by a set of sensor nodes and the topology of this set, which represents the communication links between the sensor nodes. Further on, the sensor nodes, which build the network, are defined by their collected data and their interaction capabilities with the environment or themselves. All this knowledge allows constructing a virtual representation, which can be used to represent an arbitrary wireless sensor network.

Figure 1 shows an UML diagram of the proposed virtual representation of a wireless sensor network by the presented CoMaDa framework in this paper. The three main parts of this representation are the classes

- `WSN`,
- `Node`, and
- `Topology`.

The class `WSN` maintains a set of `Node` objects, which represents the nodes building the network, as well as a `Topology` object, which represents established communication links between the nodes.

The class `Node` can be seen as the most important class of this representation. It grants access to all the capabilities the network provides. Besides offering fields for storing an identifier and some additional information (`metadata` for storing information about the type of the node and `tags` the user can define for better identification), it offers three important fields: `sensordata`, `actions` and `events`. These fields build the interface to the capabilities the node offers.

The node's collected data can be accessed by `sensordata`, which represents a set of `SensorDatum` objects that defines the type and the value of the measured data. Objects of this class also define the unit the datum was measured in. Besides of the sensor data a sensor node might also communicate arbitrary additional information. Letting the node define a set of `events` that may occur and additionally offer the user the possibility to subscribe to these allows the nodes to communicate such information. Events that may occur at a specific node are

possibly that the node has updated its measurements, that it runs out of energy soon, or any other information it wants to communicate. Such events may not only occur at node level but the wireless sensor network as whole, being an observer of all its nodes, may also want to communicate information. Therefore, the class `WSN` also offers the possibility to subscribe to events. Global `WSN` events may include notifying that the topology has changed or a node was suddenly lost. Each event is defined by a simple class, which holds some information about the event itself and about each time an event occurs. Users are notified by calling a callback function that was passed when the user subscribed to the event. An example for such an `Event` class is shown in Figure 3a.

```
class NodeLostEvent
    extends Event {

    String id;

    @EventParam
     Node lostNode;

    @EventParam
     Date timestamp;
}
```

(a) Example Event

```
class SwitchLightAction
    extends NodeAction {

    String id;

    @ActionParam
     boolean turnOn;

    @ActionResult
     boolean success;
}
```

(b) Example Action

Fig. 3: Code example for `Event` and `Action` implementations

As a consequence of the before described setup the user would be able

- To access the structure of the network,
- To access the collected data, and
- To react to events occurring within the network.

All together the user is able to read all the data a wireless sensor network offers. As already mentioned earlier, a sensor network may be able to do more than simply collecting data. It may offer the possibility to let it interact with its environment, or at least it may offer a possibility to control its behavior. Therefore, it is necessary to also be able to write to the network besides just reading from it.

For this purpose, the CoMaDa framework offers the concept of actions. Each `Node` defines a set of `actions`, which defines the ways the node can be configured at runtime and which also offers an interface to the interaction capability of the node to its environment. Since sensor nodes are very constrained devices, one can assume that the actions offered by a node are limited to rather simple processes, which can be modeled by taking a set of input parameters and returning a result. A simple example for such an action, which should command a node to manipulate the light switch it is connected to, is shown in Figure 3b. A user can declare such a class for each different action a node is capable of doing and add it to its set of `actions`.

The concept of `events` and `actions` gives the user an easy-to-use tool to model the capabilities of his network. Since `actions` and `events` are only static interfaces to their respective functionality, an application would need to know about these interfaces at the time the application is designed in order to be able to use them. This means that applications were not able to dynamically adapt to new types of nodes with new capabilities. In order to circumvent this problematic and in order to allow applications to use `events` and `actions` in a dynamic fashion, the CoMaDa framework requires those to be decorated with reflection annotations, which can be evaluated at runtime. This concept is shown in Figure 3b. The parameters and result are annotated with `@ActionParam` and `@ActionResult` respectively, which allows an application to extract the necessary information the user has to enter in order to perform the actions of a node at runtime.

Section IV shows how this information is used to dynamically adapt CoMaDa to arbitrary networks in order to offer the user a graphical interface to nodes the framework has never seen before. But this only demonstrates the basic concept of annotations. Additional, more detailed annotations, which further describe the semantics of an `action`, are imaginable. For example, a node could use these to communicate that some of its `actions` can be used to deactivate power intensive, optional features or to set the node into a power saving mode. This means that external modules could enforce power saving policies even in networks with nodes they have never seen before, and, therefore, would have had no idea of how to interact with these without being able to dynamically extract information about these using the annotations. Using the concepts of `events` and `actions` the user can reflect the functionalities of nodes in the virtual representation by simply designing own `Node` class derivatives, providing information about offered `events` and `actions` in the respective `events` and `actions` fields.

```
// NodeCollection nc;  <-- provided by WSN

nc.filter("#node1 > *");
   // selects all nodes that are linked to #node1
nc.getNodeById("node1").filterByOutgoingLinks();
   // does the same
   // but allows the compiler to check the syntax

nc.filter("tag:light,tag:kitchen")
  .performAction(new SwitchOffLightAction());
   // selects nodes tagged as "light" and "kitchen"
   // and let's them perform "SwitchOffLightAction"
   // => switches off the lights in the kitchen
```

Fig. 4: Code example for node filter feature

The last part of the virtual representation to discuss is the class `NodeCollection`, which is used by the class `WSN` to organize its assigned `Node` objects. `NodeCollection` is simply a collection of `Node` objects that offers the possibility to react to events and perform actions on a whole set of nodes by simply offering and forwarding the methods of the class `Node` for the whole collection. Additionally, it offers the possibility to filter the collection by simple patterns as shown in Figure 4, giving the user a powerful tool to select and interact with groups of nodes.

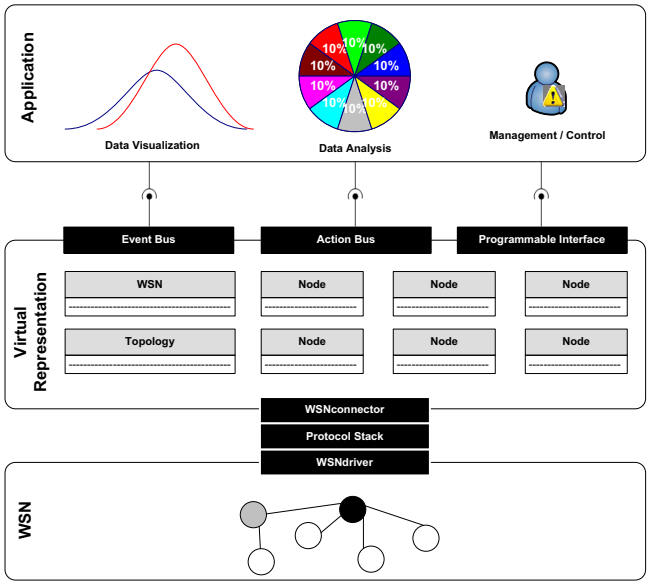

Fig. 5: General structure of the framework CoMaDa

The presented virtual representation allows an user to efficiently design applications interacting with arbitrary wireless sensor networks, without having to specialize in a specific network architecture/platform by simply relying on this additional abstract layer.

As already mentioned above the translation between the actual network and its virtual representation is done by a driver that highly depends on the used network architecture and platform. Figure 5 shows an overview of all the framework's parts and their interaction. The actual network is shown on the bottom and as can be seen, it is connected to its virtual representation by three components:

- *WSNdriver*,
- Protocol stack
- *WSNconnector*.

The component *WSNdriver* provides access to the basic functionalities of the network, managing the communication from the network to the framework and vice versa. This part only needs to be exchanged when the nodes are run with a different platform/operating system. The *WSNdriver* forwards the received data to the protocol stack. The protocol stack is responsible for the support of the different communication protocols used. After the data is translated it is passed on to the *WSNconnector* that finally is responsible for creating and updating the virtual representation. As long as no home-cooked platforms and protocols are used, the implementations of the respective driver and protocol modules can be shared across users. So in most cases the user does not need to bother implementing those when switching to a new platform, but he can simply use existing implementations. In contrast to that, the *WSNconnector*, which is responsible for the virtual representation of the network, needs to know which representation (which class) to use for which node. Since a wireless sensor network is very dynamic (nodes may be added, nodes may be removed all the time) it cannot be modeled statically and the user has to decide how to communicate and recognize the functionalities his nodes offer. The user could either include this information in a protocol, letting the nodes communicate this information themselves, or he could simply use a hard naming scheme. In this case the user could simply distinguish different kind of nodes offering different kind of functionality by the individual IDs of the nodes. Whatever way the user chooses, he has to adapt the *WSNconnector* and manually tell it what nodes shall be reflected by which class. Using the concept of *WSNdriver*, protocol stack and *WSNconnector*, applications that were designed to operate on the previously discussed virtual representation can easily be adapted to different networks and types of nodes with the following steps:

1) Exchange the *WSNdriver* if a new platform/operating system is used.
2) Exchange the *protocol stack* if different communication protocols are used.
3) Design `Node` class derivatives for the new nodes (in order to reflect new *events* and *actions*).
4) Adapt the *WSNconnector* so the new nodes are recognized.

In most of the cases only a smaller subset of the above mentioned steps are necessary since the user rather designs new type of nodes or experiments with new protocols than redesigning the whole network.

IV. Proof of Operability

The graphical user interface of CoMaDa, as described in references [1] and [5], is implemented on the server, which has unlimited resources, rendering energy consumption and memory evaluation unnecessary (cf. Figure 2). Therefore, the presented evaluation is a proof of operability of an application implemented on top of CoMaDa, the previously mentioned GUI.

CoMaDa was tested in an office scenario integrated in the project AutHoNe, which deals with autonomic home networking questions [10]. The detailed reasons for choosing an office scenario for application purposes is given in Section IV-D. The used sensor hardware was the following, which all worked under the operating system TinyOS [11]:

- IRIS nodes with sensors for temperature, light, and sound [6].
- TelosB nodes with sensors for temperature, humidity, and light [6], [7].

The authors used hardware of different vendors in order to show the compatibility of the GUI and its independent applicability. The following sections show a proof of concept for the GUI tasks introduced in Section III: (1) Configuration of network components, (2) visualization of network status, and (3) data import/export.

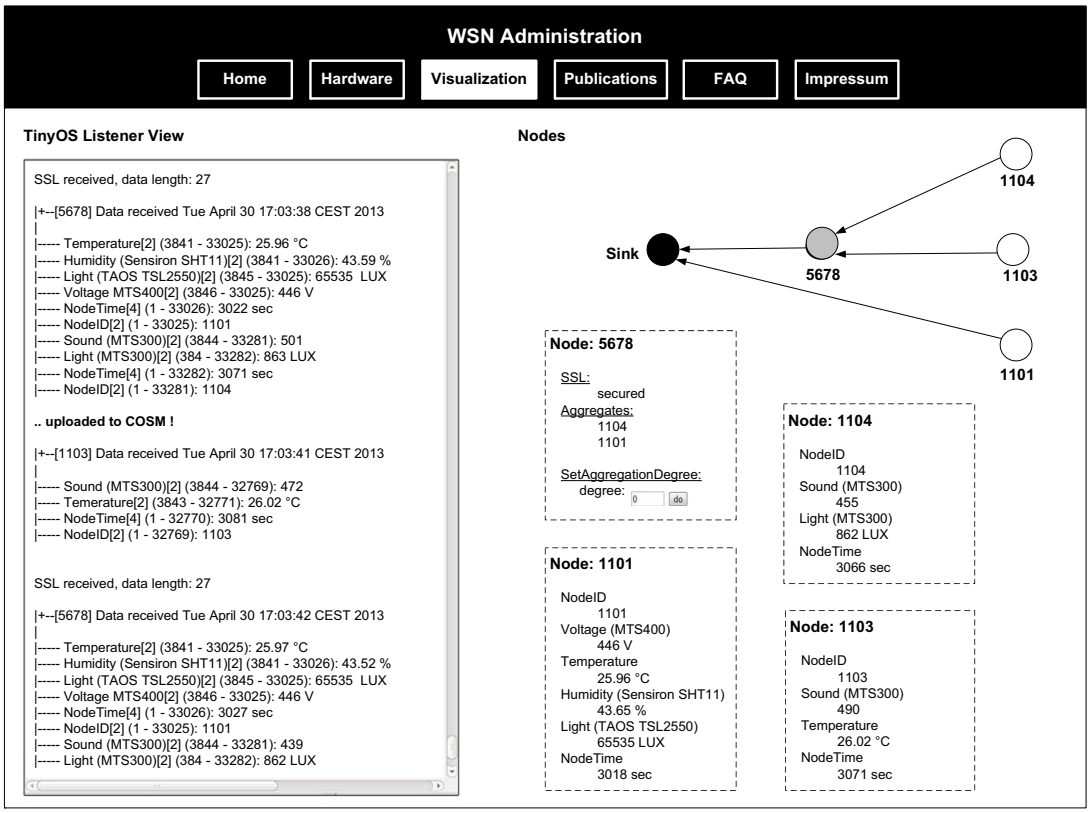

Fig. 6: Extended visual feedback of a wireless sensor network in real time [1]

A. *Configuration of Network Components*

The developers of CoMaDa decided to design it independent of the used hardware but at the same time integrated operating system depending parts (e.g. IPBaseStation from TinyOS), which allow a basic support for the operating system TinyOS. The established GUI for configuration tasks supports the programming of the required base station as well as the measurement devices. For the latter the user is able to choose between different pre-installed setups. Those setups depend on the chosen hardware and include specifications for platform type, sensor board type, TinyOS options, and node functionality. Platform types include all platforms, which are currently supported by TinyOS, such as IRIS, MICA, Imote, and TelosB. Depending on the platform structure different sensor are included or can be attached via UART to it and can be specified via the GUI. TinyOS offers different functionalities per default (e.g. routing protocol BLIP), which can be activated manually. Finally, the user must specify the performed node functionality (e.g. IPBaseStation, individual algorithms). Here the user can decide whether a pre-defined algorithm (e.g. IPBaseStation) should be installed or a personal algorithm, where the user must specify the location of the source code. With this degree of configuration as described, the user is able to configure each node in the wireless sensor network individually. [1], [5]
In order to ensure unique identities, the user has to choose an individual ID for each node in the next step. Then the user can compile the chosen code with his specifications. After the compilation the user received the required RAM and ROM size for the chosen configuration. With these facts in mind the user plugs in the appropriate sensor node and programs it. [1]
If the user wants to deploy a wireless sensor network supporting different functionalities by individual nodes the above steps can differ during configuration period. An example can be a wireless sensor network where some nodes are only data collectors with different active sensors and other sensor nodes, which perform in-aggregation.

B. *Visualization of Network Status*

After configuration of the network components the user activates the wireless sensor network. First, the sensor node, called sink, performing the program IPBaseStation is attached to the USB port of the gateway. This sensor node is a kind of agent between the wireless and the wired infrastructure of the complete system. The program allows the serial protocol to pass 802.15.4 frames in the systems, which makes the whole system interesting for systems dealing with the idea of Internet of Things (IoT) [12] and support IP communication between components [11].
Second, the user changes to the option IP tunnel where the used USB port is specified and starts the tunnel application, which is required by IPBaseStation [11]. The user can now activate individually each component of the wireless sensor network, which was configured before. In the currently open environment

in the GUI the user receives direct feedback of the running system. The tunnel traffic is visualized including activated and transmitted sensor nodes as well as received packets in raw format at the sink. New deployed sensor nodes are integrated in this visualization when they are recognized by the tunnel application, which happens when the first packet its received after sensor booting. If the wireless sensor network should be shut down the user can deactivate the IP tunnel manually by pushing the 'stop' button.
Usually in wireless sensor networks routing algorithms are integrated in order to handle the packet transmission with in the network correctly and transmitting them on an optimal route from source to sink. This routing structure is not visualized in the visualization option of the tunnel. Sometimes the user is interesting in this information as well, thus an extended visualization option was integrated in CoMaDa. Therefore the user must switch through the submenu to the option network status. Here the user can view the routing tree and the current state of all the nodes of the deployed sensor network in real time. Figure 6 shows an example for such an extended view of a wireless sensor network, which consists of four sensor nodes and the sink (=Base). On the left side of the visualization the processed tunnel traffic is illustrated. The received data at the tunnel is used as an input for the class *WSNDriver*, which forwards it to the protocol-stack for further translation. Here the user is now able to see what information each packet includes and from which node it was received. All this information is shown on the right side bottom side in a light modified version as well. Additionally, on the right upper side the currently used routing tree is illustrated including the sensor nodes with their individual IDs. Below the routing tree each node is shown in its current state, including its current sensor values and any available meta-data. In this example the wireless sensor networks consists of three data collectors with IDs 1104, 1103, and 1101. The sensor node with ID 5678 supports in-network aggregation in the system and uses SSL to encrypt the communication between itself and the sink [13] (instead of UDP, which is typically used in wireless sensor networks [9], [8]). Node 5678 aggregates the individual packets of nodes 1104 and 1103 without any modification. This operation is known under the term message aggregation. The node 1101 currently transmits its packets directly to the sink. [1], [5]

As can be seen in Figure 6, the GUI dynamically adapts to the currently attached network. All data that is available about the individual nodes gets visualized. Even the aggregator (node 5678), which uses a SSL secured transmission connection, is integrated and viewed as any other node due to the abstract representation of the network. In addition to that, the aggregator was defined to offer a method that influences its aggregation behavior and, therefore, is internally represented by a different class, which offers access to this functionality by declaring an appropriate `Action` called `SetAggregationDegree`. The GUI uses this information and displays a text-box and a corresponding button in order to allow the user dynamically trigger this function. In this visualization the edges in the routing tree display the currently used routes between the nodes. In the background each sensor node stores alternative routes as well, but those a not displayed in the visualization yet. If the deployed wireless sensor network is large scaled the complexity rises and the user might lose the overview. Therefore, the authors added an additional feature to this visualization: The user can click on the sensor node with its individual packet information below. As a consequence the corresponding sensor node and it ingoing and outgoing routes in the routing tree are highlighted in order to find the sensor node more quickly. [1]

C. Data Import and Export

The amount of collected sensor data depends on the size of the network, on collected values, and on collection intervals. As a consequence this data amount can be huge. Today it is common to analyse the data by plotting them. Different tools are available, which offer different plotting techniques (e.g. line, point, step diagrams) and can be divided into offline and online tools. In the case of offline tools (e.g. Matlab) the collected data must be stored on the server for further process. The online tools (e.g. COSM [14]) allow the user to upload data in real time and to display them directly. In order to support both options the established GUI includes data import and export functionality.
All received data is stored locally on the server in tab-separated files, which can be imported to offline analysis tools (e.g. Matlab) and be analysed and plotted depending to the users requirements. The final plot is currently not imported back into the GUI. Thus, the user has to view it outside the GUI on its specific terminal or browser.
In the case of online tools, such as COSM [14], the received data from the wireless sensor network must be converted to the required data format, which is in case of COSM the JavaScript Object Notation (JSON) language, before the data can be exported to the tool. The COSM module implicitly does this and the user only has to specify which individual sensor node of the system should be exported and visualized. Therefore, the user creates feeds (models used by COSM to group data-streams) for each node he wants to export and the COSM-module automatically translates between CoMaDa's virtual representation and COSM's feed architecture. Per default all data included in the packet is displayed via COSM (e.g. Node ID, time, light, sound). The resulting visualization is imported to the GUI and visualized to the user. Figure 7 illustrates such a visualization result where the user has discarded the plotting of time and Node ID, because it is redundant to the information shown in the left part of the figure including all default data. As a result in this example only the plots for light and sound are visualized. Here the user has also the possibility to regulate the time scale of the plots (here: 3 hours). Periodically in the background of the GUI the plots are stored in order to view them offline again. If the user shuts down the wireless sensor network completely, the plot is stored in total over the whole plotting time for each sensor node, which was plotted. [1]

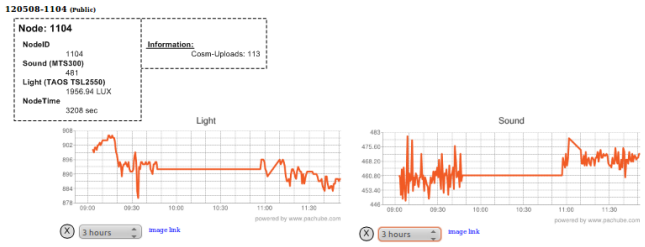

Fig. 7: Imported COSM plot [1]

D. Operability evaluation

As mentioned in the beginning of Section IV an office scenario was chosen for operability proof. An office scenario was a reasonable possibility to test the GUI based on CoMaDa on behalf of the framework itself with all its supports, because it allowed the authors to test the GUI on different scaled networks sizes (3-30 nodes), different distances between sensor nodes and to the sink, and during different experimental durations (5 min - 3 days). Additionally it was possible to test the GUI in an environment, which included different architectural structures (e.g. glass, concrete and wooden walls, metal infrastructure, influence of different radio settings).

The developed GUI was tested extensively and scaled well on different setting. Currently the only requirement is that the sampling rate should be chosen appropriately to the network size. This means that if the network consists of more than 15 sensor nodes the sample rate should be set to higher than 25 seconds in order to avoid packet lost. Another drawback is currently the number of possible active connections to COSM, which is limited to five at the same time. This means it is only possible to export five nodes at the same time. If a display of one node is shut down the user can directly open a new connection to display another one. This exporting drawback does not influence the visualization of live stream and routing as shown in Figure 6.

Independently of the currently supported operating system TinyOS further investigations exist to support the operating system Contiki in the future. Additionally, more analysis tools will be integrated in the GUI. Those extensions of the GUI will be possible with less manual input due to the modular structure of the GUI as described in Section III.

V. Conclusion

As motivated at the beginning of this paper existing frameworks and GUIs for wireless sensor networks focus on simulation scenarios or are bound to specific hardware/platforms. Therefore, this paper deals with the realization of the flexible and adaptable framework CoMaDa, which supports management tasks for different hardware types and visualization of collected data, as well as feedback about the system status in real time. The flexibility of CoMaDa is gained by a virtual representation of the real wireless sensor network, which offers independency of the application, algorithms, and hardware vendors. In addition, the framework was built in a modular structure, which offers the user the opportunity to modify CoMaDa to his/her requirements by adding new functionalities (e.g. adding support for new nodes, new protocols, new operating systems) with fewer overheads. With its adaptive and modular structure, CoMaDa makes a great base for building applications that interact with wireless sensor networks but are not bound to specific platforms or setups.

The authors believe that the presented framework can charm developers to design and implemented their applications with no specific target platform in mind, so the community could benefit from these regardless of their preferred platform. This would lead to the benefit that the set of available tools is no reason for deciding for a specific platform anymore. For testing purposes in individual applications the source code of the framework including the GUI is available with MIT-License under reference [15]. The authors will be grateful for every user feedback in order to make the framework more flexible and user friendly.

References

[1] C. Schmitt, “Secure Data Transmission in Wireless Sensor Networks,” Ph.D. dissertation, Technische Universität München, Germany, Department of Computer Science, Network Architectures and Services, July 2013.

[2] B. Parbat, A. K. Dwivedi, and O. P. Vyas, “Data Visualization Tools for WSNs: A Glimpse,” in *International Journal of Computer Applications*. Foundation of Computer Science, 2010, vol. 2, pp. 14–20.

[3] I. Crossbow Technology, “MoteView Users Manual ,” Crossbow Technology, Inc., Tech. Rep. 7430-0008-04, November 2006.

[4] H. Sundani, H. Li, V. K. Devabhaktuni, M. Alam, and P. Bhattacharya, “Wireless Sensor Network Simulators A Survey and Comparisons,” *International Journal of Computer Networks (IJCN)*, vol. 2, no. 6, pp. 249–265, 2011.

[5] A. Freitag, “Framework Development for Wireless Sensor Networks facing Configuration and Information Exchange/Export Tasks,” Bachelor Thesis, Department of Computer Science, Technische Universität München, Germany, 2011.

[6] Crossbow Technologies Incorporation, http://www.xbow.com/, 2013.

[7] Advantic Sistemas Y Servicios S.L., http://www.advanticsys.com, 2013.

[8] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “A Survey on Sensor Networks,” *Communications Magazine, IEEE*, vol. 40, no. 8, pp. 102–114, August 2002. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1024422

[9] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. Wiley-Interscience, 2007.

[10] “Autonomic Home Networking - Project Homepage,” http://www.authone.de, 2013.

[11] “Tinyos homepage,” http://www.tinyos.net/, 2013.

[12] I. Strategy and P. Unit, “ITU Internet Reports 2005: The Internet of Things,” *Geneva: International Telecommunication Union (ITU)*, 2005. [Online]. Available: http://www.itu.int/internetofthings/

[13] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, “DTLS based Security and Two-Way Authentication for the Internet of Things,” *Ad Hoc Networks*, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870513001029

[14] COSM Connect to your world, http://www.cosm.com, 2012.

[15] C. Schmitt and A. Freitag, “Sourcecode for GUI Framework,” http://www.net.in.tum.de/de/sandkiste/wireless-sensor-networks/, 2013.