Towards Plug-and-Play Functionality in Low-Cost Sensor Network

Rajib Rana ^{#1}, Neil Bergmann ^{#2}, Jarrod Trevathan ^{*3}

[#] *ITEE, University of Queensland, Australia*

¹ *rajeevrans@gmail.com*, ² *neil.bergmann@itee.uq.edu.au*

^{*} *eResearch Centre, James Cook University, Townsville, Australia*

³ *jarrod.trevathan@jcu.edu.au*

Abstract—Plug-and-play functionality in sensor networks is becoming inevitable to make network configuration and reconfiguration flexible and cost effective. The IEEE 1451 standard brings forward the concept of Transducer Electronic Data Sheet (TEDS), which is a memory device attached to the transducer, storing detailed information to configure the corresponding transducer. In this paper we share our experiences of using TEDS to enable plug-and-play functionality within a low-cost sensor network.

I. INTRODUCTION

Sensor networks are gaining increasing interest for monitoring numerous environmental phenomena, from air quality and noise pollution in the surrounding environment [1] to temperature and salinity in the marine environment [2], [3]. However, in order to maximize the benefits from the sensor network, one key challenge need to be addressed is, how to enable the plug-and-play functionality in the sensor network? Sensor networks typically consist of heterogeneous sensor nodes and devices. In order to add a new type of sensor node, the whole system need to be configured so that the sensor reading can be parsed at the user end. This problem becomes extreme in the marine sensor network, where configuration of every new sensor may involve sending divers underwater. A plug-and-play system is therefore inevitable where the new nodes can be automatically detected and configured by the network allowing instant use.

The IEEE 1451 standard [4] defines transducer interface standards, which describe a set of network-independent communication interfaces for connecting transducers (sensors or actuators) to microprocessors. One of the key elements of these standards is the Transducer Electronic Data Sheet (TEDS). The TEDS is a memory device attached to the transducer, which stores transducer identification, calibration, correction data, and manufacturer-related information. In particular, the goal of the IEEE 1451 family of standards is to allow the access of TEDS through a common set of interfaces while the transducers are connected to systems or networks via a wired or wireless means, and thus enabling the automatic configuration of the transducer to the system.

In this paper we investigate the use of the IEEE 1451 standard to enable the plug-and-play functionality in the sensor network. A number of projects have been conducted in the past where researchers have attempted to use IEEE 1451 within wireless sensor networks [5]. However, the work in [5] lacks

guidelines explaining how similar system can be developed on different platforms. Furthermore, they use TelosB sensor nodes, which are quite expensive. In this paper we describe the details of writing TEDS for the inexpensive and popular Arduino UNO [6] platform. In the past Arduino has been used [7] to model low-cost prototyping platform, however, enabling plug-and-play functionality through Arduino has not been explored yet. We also demonstrate within a wired network how TEDS can be accessed from the Arduino platform. In the end we share valuable lessons that we have learned throughout the experimental work conducted in this paper.

This paper is organized as follows. In the next section (Section II) we describe the network architecture used in this paper. Then in the following section (section III) we describe TEDS in details. In Section IV we describe the implementation details of TEDS in Arduino UNO platform and in Section V we evaluate our implementation. Finally, we conclude in Section VI.

II. ARCHITECTURE

We consider a sensor network as shown in Fig. 1. Below we provide a component-wise description of the architecture.

Fig. 1: Sensor network architecture.

A. Transducer Interface Module (TIM)

In our architecture the TIM includes an Arduino UNO, a thermistor (connected to the UNO's analogue input) and a TEDS stored in solid state memory. A Block diagram of TIM is illustrated in Fig. 2.

Fig. 2: Block diagram of TIM

1) *Arduino UNO*: The Arduino UNO uses a ATmega328 microcontroller. It has 14 digital input/output pins, 6 analog inputs and a 16 MHz crystal oscillator. Note that there are six analogue input pins in the UNO board. Therefore, at least six sensors can be attached to the UNO board.

Fig. 3: a) Thermistor connected to a voltage divide circuit. b) Arduino UNO connected to a thermistor.

2) *Thermistor*: A thermistor measures the temperature in terms of change in resistance. We use the NTC Thermistors that comes with the Arduino UNO inventor's kit. Note that the analog input pins of Arduino cannot measure resistance, therefore, the change in the resistance cannot be measured directly. A simple way to convert the resistance to voltage is to connect the resistance in a voltage divider circuit as shown in Fig. 3(a). In Fig 3(b) we demonstrate the physical connection of an Arduino UNO and a thermistor.

The voltage divider circuit has two registers in series. The register in the top is the thermistor which has a variable resistance R_t . The register (R) at the bottom is a fixed register. A voltage V_s is applied to the circuit and the output voltage V_o is read at the analog input pin. Using V_o , V_s and R , the value of the variable resistance R_t can be computed as follows:

$$R_t = R \left(\frac{V_s}{V_o} - 1 \right) \quad (1)$$

The value of the unknown resistance R_t can be computed from (1). Given the value of R_t , temperature value can be computed using the following Steinhart-Hart equations (2) [8].

$$\frac{1}{T} = c_1 + c_2(\ln(R_t)) + c_3(\ln(R_t))^3 \quad (2)$$

Here, c_1 , c_2 and c_3 are constant for a given thermistor. The values of these constants can be found in the thermistor data sheet. However, they can also be determined empirically. This involves simple laboratory tests, where resistance of the thermistor at different temperatures are computed. Then values of c_1 , c_2 and c_3 are adjusted so that (2) fits the data points

(temperature-resistance mappings). Details of this procedure can be found in [9].

B. Hub

The hub is relatively powerful compared to the TIM. In our current prototype implementation we use a PC as hub, which connects to the TIM using an RS232 port.

III. PLUG AND PLAY THROUGH TEDS

Generally speaking, TEDS is the electronic version of the data sheet that is used to configure a sensor. TEDS brings forward the concept that if the data sheet is electronic and can be readily accessed upon sensor discovery, it would be possible to configure the sensor automatically. This is analogous to the operation of plugging a mouse, keyboard, or monitor in the computer and using them without any kind of manual configuration.

The IEEE 1451 standard defines TEDS as a collection of multiple sections tailored together to form a complete TEDS. TEDS typically resides within the EEPROM of the Transducer Interface Module or TIM. In our implementation, the Arduino UNO board is used as TIM. Some of the sections of the TEDS are mandatory and some of them are optional. The mandatory sections are Meta TEDS, TransducerChannel TEDS, User's Transducer Name TEDS, and PHY TEDS. The optional TEDS are calibration TEDS, Transfer Function TEDS, Frequency Response TEDS, etc. Below we provide a brief description of different types of TEDS that we have implemented in our test configuration.

A. Meta TEDS

The Meta-TEDS is a mandatory TEDS. It makes available at the interface all information needed to gain access to any TransducerChannel, and information common to all TransducerChannels.

B. Transducer Channel TEDS

This is a mandatory TEDS. The TransducerChannel TEDS makes available at the interface all of the information concerning the TransducerChannel being addressed to enable the proper operation of the TransducerChannel.

C. User's Transducer name TEDS

The user's transducer name TEDS mainly contains the model number of the TIM.

D. Calibration TEDS

The calibration TEDS provides the calibration constant required to convert a sensor's output to an engineering unit. In particular, it provides constants to perform two type of conversions: 1) converting raw input data to appropriate output (e.g., in our case it provides constants to convert the resistance to temperature) 2) converting from sensor unit to the SI unit (e.g., in our case converting Celsius to Kelvin). Assuming a linear calibration curve following equation (3) is used for the conversion from the sensor unit to the SI unit.

$$y = mx + b. \quad (3)$$

Here m is the slope (SI unit), b is the intercept (SI unit) and x is the number produced by the correction process (sensor unit).

E. PHY TEDS

The PHY TEDS depends on the medium used to connect the Hub and the sensors. In our implementation we use 1451.2-RS232. Note that there are also other interfaces defined in 1451.2, such as, UART and USB. In addition, IEEE 1451.5 defines wireless interfaces such as, 802.11 (WiFi), 802.15.1 (Bluetooth) and 802.15.4 (ZigBee).

IV. TEDS IMPLEMENTATION FOR ARDUINO UNO

In this section we describe the details of the TEDS implemented for Arduino UNO. We implemented one channel and connected a thermistor to the Arduino UNO. Different sections of the TEDS are described in the following.

A. Meta TEDS

Meta TEDS consists of the following fields:

1) *Meta TEDS ID*: Values used for Meta TEDS ID are shown in Table I. Meta TEDS has type 03 and the total length of the field is 04 (as shown in Table I). Therefore, in Type-Length-Value (TLV) format this becomes: 03 04 00 01 01 01

2) *Universal Unique Identification Data (UUID) type Field*: The UUID is an identification filed of TIM whose value is unique in the world. It consists of four subfields, 1) location of manufacturer, 2) manufacturer's field 3) year field 4) time field. Note that it is not mandatory that these fields need to describe accurate location or time. However, the manufacturer needs to choose these values in such a way so that no other manufacturer can claim the same UUID for a TIM.

In the following we mention the values that we have used for various subfields.

Location of the Manufacturer: The location of the Arduino UNO manufacturer is Via Romano, 12 10010 Scarmagno, Italy. This has the GPS coordinates Lat 45.384458N (163384.04 arcseconds), Lon 7.840858S (28227.08 arcseconds).

Manufacturer's Field: If there is only one manufacturer at the previous location, this field is 0, otherwise it should reflect the number of manufacturers.

Year Field: This field represents the year the board was built. The Arduino UNOs we used were manufactured in 2011.

Time Field: This value should be chosen such that the combination of all fields in Meta TEDS ID is unique. The date of manufacture of the board was 1st January, 2011. Time formulation of the field is bit arbitrary. We formulate this field in the following manner: 1st January is the first day of the year and the board was 120th production of the day. So the time field we chose to be $1 \times 1000 + 120 = 1120$.

Different field values (in Binary) of UUID are shown in Table II. Realigning (group by 8 bits) and converting to octets yields the following UUID format: F1 C0 1B 90 81 F6 C0 04 60. Type of the UUID is 04 and there are 10 octets, therefore the length is 0A. Finally, UUID in TLV format is as follows: 04 0A F1 C0 1B 90 81 F6 C0 04 60.

TABLE II: UIUD Fields

Field	Value	Binary Value	Bits
N/S	N	1	1
Latitude	163384.04	00100111111000111000	20
W/S	E	1	1
Longitude	28227.08	00110111001000011	20
Manufacturer	0	0000	4
Year	2011	011111011011	12
93 Date/Sequence	1120	0000000000010001100000	22

3) *Operational Timeout*: This is the time interval in seconds. If there is no reply by this interval, the operation is assumed to be failed. Later in this paper we have shown that the maximum time required by any TEDS operation is 0.1s. We therefore choose a time larger than 0.1s as our operational time out. We use 0.5 seconds as the operational time-out. Converting it to F32 floating point and then to Hex: 3F 00 00 00. Field type is 10 (0A in hex) and length of the field is 04. Therefore, in TLV operational timeout becomes: 0A 04 00 00 00.

4) *Self-test time-out*: There is no self-test implemented, therefore this field is zero. Type of this field is 12 (0C in hex) and length is 01. In TLV this field becomes: 0C 01 00.

5) *Number of Transducer Channels*: We implement only one channel. Therefore the value of this field is 1. The data type is unsigned 16, therefore, require 2 octets. Field type is 13 (0D in hex). Therefore, in TLV it becomes: 0D 02 00 01.

B. Checksum

The checksum is computed by adding the octets of the data and the length fields and taking ones complements. This can be described by the following formula

$$\text{checksum} = 0xFFFF - \sum_{i=1}^{\text{TotalOctets}-2} \text{TEDSOctet}(i) \quad (4)$$

Here TotalOctets contains the total number of octets including the checksum. We compute our checksum for Meta TEDS to be FA C5.

C. TransducerChannel TEDS

The TransducerChannel TEDS has the following fields.

1) *TransducerChannel TEDS ID*: TransducerChannel TEDS ID has the same fields as of Meta TEDS ID (see Table I) except the value of the field class is 03. In TLV format it is given by: 03 04 00 03 01 01.

2) *Calibration Key*: Our TIM will provide calibration information that is required by the application. The field type is 10 (0A in hex) and the length of the field is 01. In TLV format it becomes: 0A 01 01.

3) *TransducerChannel type key*: Our TIM has one sensor channel, so the transducer type is 0. Note that value of this field is 0 for sensor, 1 for actuator and 2 for event sensor[4]. The field type is 11 (0B in hex). In TLV format it becomes: 0B 01 00.

TABLE I: Meta TEDS ID

Field	Contents	Functions
Family	00	Identifies the member of the IEEE 1451 family of standards that defines the TEDS.
Class	01	Identifies the TEDS being accessed.
Version	01	Identifies the TEDS version.
Tuple Length	01	This field gives the number of octets in the length field of all tuples in this TEDS except this tuple.

TABLE III: Physical Units

Field	Description	Type (Hex)	Value(Hex)
1	Physical unit interpretation	50 (32)	0(0)
2	(2*exponent of radians)+128	51 (33)	128(80)
3	(2*exponent of steradians)+128	52 (33)	128(80)
4	(2*exponent of meters)+128	53 (33)	128(80)
5	(2*exponent of kg)+128	54 (33)	128(80)
6	(2*exponent of seconds)+128	55 (33)	128(80)
7	(2*exponent of amperes)+128	56 (32)	0(0)
8	(2*exponent of kelvins)+130	57 (33)	130(82)
9	(2*exponent of moles)+128	58 (33)	128(80)
10	(2*exponent of candelas)+128	59 (33)	128(80)

4) *Physical Units*: The description of the physical units are shown in Table III. Since we only use one temperature sensor, field 8 is used and Field 1 is used regardless. Type of physical unit is 12 (0C in hex). Therefore, in TLV it is given by: 0C 06 32 01 00 39 01 82.

5) *Design Operational Lower Limit*: Arduino UNO has an operational lower limit -40°C . The operational lower limit needs to be specified in Kelvin. The kelvin value is first represented in F32 floating point then in hex: 43 69 26 66. The field type is 13 (0D in hex). Therefore, in TLV the field is given by: 0D 04 43 69 26 66.

6) *Design Operational Upper Limit*: Arduino UNO has an operational upper limit 85°C . The field type is 14 (0E in hex). In TLV this is given by: 0E 04 43 B3 13 33.

7) *Self-test key*: Arduino UNO has no available self test. Therefore, this field is set to 0. The field type is 16 (10 in hex). Therefore, in TLV it becomes: 10 01 00.

8) *Sample Definition*: Sample definition consists of three subfields.

Data Model Our TIM outputs resistance of the thermistor, which can be represented using single precision real numbers (4 bytes). According to the standard (Table 52 in [4]) a single precision real number is represented using 1. So, the value of this field will be 01. The field type is 40 (28 in hex). In TLV: 28 01 01.

Model Significant Bits When the data model is a single precision real number, the value of this field is the number of bits of the transducerChannel's signal converter. The analogue to digital converter on Arduino has 10 bits of resolution. Therefore, the value of The value of this field is 10. The field type is 42 (2A), so the octet pattern in TLV is: 2A 01 0A.

Final Octet format of Sample Definition The total length of these three subfields is 6 and the type of the final field is 18 (12 in hex). Therefore, the sample definition field in TLV: 12 06 28 01 01 2A 01 0A.

9) *TransducerChannel Update Time*: We have configured the sensor to run in a continuous mode. This mode uses

multiple buffers to store the data. When all the buffers are full data is being transmitted to the Hub from the oldest buffer. Newly acquired data is then stored in the empty buffer. The sampling interval is also programmable. For a sensor running in continuous mode with a programmable sampling interval, the TransducerChannel update time is irrelevant and set to zero. The field type is 20 (14 in hex). In TLV: 14 01 00.

10) *TransducerChannel read setup time*: The read setup time represents the time between receiving a trigger by the TIM and making the data available to read. We have measured this time to be 25 micro seconds. The field type is 22 (16 in hex). This yields the following octet pattern: 16 04 37 D1 B7 17.

11) *TransducerChannel Sampling Period*: For sensor and actuator TransducerChannels, sampling period is typically limited by A/D or D/A conversion times, transducer module processor speed, and so on. We choose a value of sampling period which is higher than the read setup time. We decide sampling period to be 0.1s. The data type is F32 and the field type is 23 (17 in hex). In TLV: 17 04 3D CC CC CD.

12) *TransducerChannel read delay time*: This is the maximum amount of time between receiving a read command and the beginning the transmission of data frame. This time is same as read setup time which is 25 micro second. The field type is 25 (19 in hex) and the data type is F32. In TLV: 19 04 37 D1 B7 17.

13) *TransducerChannel Self-Test Time requirement*: Since self-test is not available for Arduino, this field can be omitted.

14) *Sampling Attribute*: This field consists of two subfields.

Sampling mode Capability This field describes whether the transducer is in free running or pre-trigger mode. This is represented by a eight bit number, where if the first bit is set, it represents that the transducer is in pre-trigger mode. If the second bit is set it denotes that the transducer is in free running mode. When the second bit is set, the value of this field becomes 2 (note that our sensor runs in free running mode). Type of this field is 48 (30 in hex). In TLV: 30 01 02.

Default Sampling Mode Since the only possible value of sampling mode is free run without pre-trigger, it is also default and can be omitted.

Final Sampling attribute Type of this attribute is 31 (1F in hex). In TLV the field is given by: 1F 03 30 01 02.

15) *Checksum*: The checksum is computed using (4), which becomes F4 B1.

D. User's transducer name TEDS

1) *User's transducer name TEDS ID*: The subfields of this TEDS field are same as Meta TEDS ID and TransducerChannel TEDS ID, except the value of the class is different, which

is 12 (0C in hex). Therefore, in TLV format the field is given by: 03 04 00 0C 01 01.

2) *Format*: In our implementation we use a text field. According to the standard the text field is represented using value 0. The field type is 4. In TLV this field is given by: 04 01 00.

3) *TCName*: This contains the model name of the TIM which is “Arduino TEDS”. Since this field is user dependent, no TLV format is required. Just the ASCII character values are used to form the field: 41 52 44 55 49 4E 4F 54 45 44 53.

4) *Checksum*: Total number of octets in this TEDS is 21 (15 in hex). The checksum is computed as FB 6A using (4).

E. Calibration TEDS

The calibration TEDS has the following fields.

1) *Calibration TEDS ID*: Similar to other TEDS, this TEDS has same number of fields in the ID. Furthermore, all the values except the value of the class field are the same as other TEDS. Class field has value 05. In TLV: 03 04 00 05 01 01.

2) *Last Calibration Date*: Our TIM was last calibrated as the part of the manufacturing process. Therefore, the manufacturing date and the last calibration date are the same, which is 1st January, 2011. The manufacturing time is 14:00 (Note that the manufacturing date and time are imaginary and made up to illustrate the TEDS formation mechanics). Total number of seconds since 1970 to the calibration date is 1293890400. Converting this to Hex and timeInstance (32 bits) yields a value 4D 1F 33 60 00 00 00 00. The field type is 10 (0A in hex) and length is 08. Therefore, in TLV: 0A 08 4D 1F 33 60 00 00 00 00 00.

3) *Calibration Interval*: The value of this field is arbitrary. Some sensors need frequent calibration where as others need it infrequently. We have configured the calibration interval to be one year. Converting to seconds and representing in hex, it becomes: 01 E1 33 80 00 00 00 00. The field type is 11 (0B in hex) and the length is 8. In TLV this field is: 0B 08 01 E1 33 80 00 00 00 00.

4) *SI unit conversion constant*: Since the calibration data is going to be shown in Celsius, we need to provide the SI unit conversion formula. The formula is Kelvin = deg C + 273.15. Here the slope is 1 and can be omitted. The value of intercept is 273.15 which is a F32 type value and its field type is 31 (1F in hex). In TLV the slope is given by: 1F 04 43 88 93 33. Finally, the SI unit conversion constant has a type 12 (0C in hex) and including the slope its length is 06. Therefore, in TLV: 0C 06 1F 04 43 88 93 33.

5) *General Conversion Method*: This is different from the SI unit conversion. For our implementation, it involves conversion from resistance to temperature. The corresponding conversion formula is given in equation (2). The calibration TEDS simply store the value of the constants c_1 , c_2 and c_3 in an one dimensional array named “set of coefficients”, which is later used for this conversion.

TABLE IV: Command Classes.

CmdClassID	Attribute name	Description
1	CommonCmd	Common commands
3	XdcrOperate	Transducer operating state

TABLE V: CommonCmd Functions.

CmdFunctionId	Command	Reply
1	Query TEDS	yes
2	Read TEDS	yes
3	Write TEDS	no
4	Update TEDS	yes

F. Commands to Access different TEDS

We follow the command structure defined in IEEE 1451.0 [4] to access different TEDS. A command message consists of destination (TransducerChannel number), command class, command function, length of message, and command related octets (offset). The IEEE1451.0 defines eight standard command classes indexed from 0-7, There are also reserved classes (8-127) and open classes to manufactures (128-255). Table IV shows the three IEEE1451.0 standard commands classes implemented in our application. The CommonCmd class is used to configure all mandatory TEDS whereas the XdcrOperate class allows reading or writing the transducer channels in each TIM.

Within each class we have implemented few functions. A list of these functions is given in Tables V and VI, respectively.

V. IMPLEMENTATION AND EVALUATION

1) *Implementation of TIM*: A TIM was implemented using C++. TEDS were stored in the EEPROM. Out of the 1K bytes of EEPROM memory space, the TEDS used up 225 bytes (Meta TEDS 31 bytes, TransducerChannel TEDS 71 bytes, User’s Transducer name TEDS 22 bytes, PHY TEDS 56 bytes and Calibration TEDS 47 bytes), which is about 22% of the EEPROM. A lookup table was created to map the TEDS commands to the corresponding actions. The lookup table was stored in the Flash memory. Each row of the lookup table had three tuples: a command, starting bytes of the memory address of the corresponding reply and length of the reply. The size of the lookup table was 252 bytes, which is 1.6% of the flash memory.

2) *Implementation of Hub*: In our prototype implementation the hub was implemented in a PC. In particular we used C++ to develop a module that can pass commands to the TIM and can parse the replies from the TIM. For example, the read TransducerChannel data command was executed by executing following three actions 1) read the data from the thermistor 2) fetch and parse the calibration constants (c_1, c_2

TABLE VI: XdcrOperate Functions

CmdFunctionId	Command	Reply
1	Read TransducerChannel	yes
2	Write TransducerChannel	yes

Fig. 4: Read channel data command.

Fig. 5: Response of Channel Read command.

and c_3 for resistance to temperature conversion and m and b for SI unit conversion) from the EEPROM and finally, 3) apply the constants the resistance reading and display the reading in the user interface.

3) *Evaluation*: In order to evaluate the system we connected one TIM with PC. We passed various commands to the TIM and evaluated the correctness of the reply messages. Due to space limitation we cannot provide examples of all the commands we have implemented. However, in Fig. 4 we provide an example of command and reply message for reading Transducer Channel Data, which will demonstrate the formulation technique of the TEDS access commands. We have implemented one channel in the transducer. In Fig. 4 the destination (0x00 0x01) means channel one (temperature sensor). The command class (0x03) means transducer operation. Finally, the command function (0x01) means read transducer channel data.

Reply of this corresponding command is shown in Fig 5. The first seven numbers (0x01 0x00 0x06 0x00 0x00 0x00 0x00) are the header information. The last two numbers (0x00 0x18) are the temperature reading. The temperature is shown in Celsius which refers to 24°C. We also provide a snapshot (see Fig. 6) of the user interface that we have implemented for reading the TEDS. The snapshot shows the reply message of the Meta TEDS read request.

VI. LESSON LEARNED AND FUTURE WORK

Based on our experiences with TEDS we believe that TEDS enabled sensors certainly pave the pathway towards

plug-and-play sensor connectivity. However, implementation of TEDS need to be situation specific. Such as, if there are 100 temperature sensors from same manufacturer deployed in a network, it is not necessary to install the full TEDS in all the corresponding TIMs, since a lot of information is common to all the sensors. If this common information can be stored in the Hub and only information unique to sensors (such as calibration constant for resistant to temperature conversion) can be stored in the corresponding TIMs, it would lead to more feasible plug-and-play systems. Currently there are two approaches of storing the TEDS: 1) the complete TEDS is installed in the TIM and 2) the TEDS can be downloaded from some storage within the network (virtual TEDS). Both of these two approaches have problems. In the first case memory can be a constraint (Arduino UNO has only 1K EEPROM), and in the second case accessing the TEDS from a remote location may incur additional network cost. Our future study includes the investigation of the partial TEDS (common information in Hub and unique information in TIM) installation on TIM, to achieve a more feasible plug-and-play sensor connectivity.

ACKNOWLEDGMENT

This work was supported in part by the Queensland Government National and International Research Alliances Program.

REFERENCES

- [1] R. K. Rana, C. T. Chou, S. S. Kanhere, N. Bulusu, and W. Hu, "Earphone: an end-to-end participatory urban noise mapping system," in *IPSN '10: Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. New York, NY, USA: ACM, Apr. 2010, pp. 105–116.
- [2] R. Johnstone, D. Caputo, U. Cella, A. Gandelli, C. Alippi, F. Grimaccia, N. Haritos, and R. E. Zich, "Smart environmental measurement and analysis technologies (semat): Wireless sensor networks in the marine environment," in *Wireless Sensor and Actuator Network Research on Opposite Sides of the Globe (SENSEI)*, Stockholm, Sweden.
- [3] J. Trevathan, I. Atkinson, W. Read, N. Bajema, Y. J. Lee, A. Scarr, and R. Johnstone, "Developing low-cost intelligent wireless sensor networks for aquatic environments," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2010 Sixth International Conference on*, dec. 2010, pp. 13–18.
- [4] "IEEE standard for a smart transducer interface for sensors and actuators - common functions, communication protocols, and transducer electronic data sheet (TEDS) formats," *IEEE Std 1451.0-2007*, pp. 1–335, 21 2007.
- [5] J. Higuera, J. Polo, and M. Gasulla, "A zigbee wireless sensor network compliant with the ieee1451 standard," in *Sensors Applications Symposium, 2009. SAS 2009. IEEE*, feb. 2009, pp. 309–313.
- [6] SmartProjects, "Arduino UNO," <http://arduino.cc/en/Main/ArduinoBoardUno>.
- [7] N. Bergmann, M. Wallace, and E. Calia, "Low cost prototyping system for sensor networks," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2010 Sixth International Conference on*, dec. 2010, pp. 19–24.
- [8] J. Steinhart and S. Hart, "Calibration curves for thermistors," *Deep-Sea Research*, vol. 15, pp. 497–503, 1968.
- [9] G. Recktenwald, "Thermistor calibration:," <http://web.cecs.pdx.edu/~gerry/class/EAS199B/howto/>.

Fig. 6: TIM Command Interface.