Low Level Communication Protocol and Hardware for Wired Sensor Networks

Jan Stepan, Jan Matyska, Richard Cimler, Josef Horalek
Department of Applied Informatics, University of Hradec Kralove, Hradec Kralove, Czech Republic.
jan.stepan.3@uhk.cz

***Abstract*—This paper describes communication protocol, which was originally designed and implemented for end point communication in smart homes. Due to protocol simplicity and architecture robustness, it is possible to use the whole system for other purposes rather than smart home control. For this purpose, the protocol uses the very common RS485 bus. Using metallic connection nowadays may look like a step back, but it is the fastest and cheapest way to integrate real hardware devices into the existing simulation projects. This paper describes the requirements for the whole hardware and software architecture, emphasizing on the descriptions of the protocol and the lowest hardware layers.**

***Index Terms*—Sensor Networks; RS485; Home Automation.**

I. Introduction

Systems for home automation consists of two parts: The first part is the control unit, and the second is the modules connected to it. Modules are various peripherals (actors, controllers and sensors) placed in a house. For example buttons, switches, lightbulbs, thermostatic heads and etc. They are connected with control unit by wires or wirelessly [1]. Communication protocols and transmitting technologies vary by specific manufacturer. Portfolio of the available modules may also vary as well as the capabilities of control unit.

Many companies worldwide offer very good to excellent smart home solutions. For examples, the iNELS [2], Loxone [3] and Fibaro [4]. These companies have extensive range of various modules to buy. However, there is a problem of architecture, in which there is no way to control home, if the control unit is broken or fail to work.

There are also many researchers who have successfully tried to make this area better. There are papers [5] [6] that describe algorithms for fall detection using cameras. [7] presents algorithm for learning inhabitant's behavior patterns and use them for better automation. Machine to machine learning networks are described in [8]. However, ways to integrate these approaches with real physical hardware are still non-existance as the commercial solutions are not open for third party usage and developing one purpose hardware for testing is costly and demanding.

For this reason, a new architecture has been designed and implemented. The main goal is to make it open for third party integration so that it is more secure than the existing commercial products with the possibility to use it in areas other than smart home automation.

As shown in Figure 1, the existing smart home architecture is expanded by adding another layer – the lowest logical layer containing nodes. This name was chosen to differentiate it from the other modules. The module stands for various types of devices according to its manufacturer. Some are equipped with microcontroller and some are not. Some communicate by analog signals and some over digital bus. Microcontrollers can be programmed and used for getting or setting data from any existing hardware components [9] [10] [11]. They can communicate with higher layers through data buses.

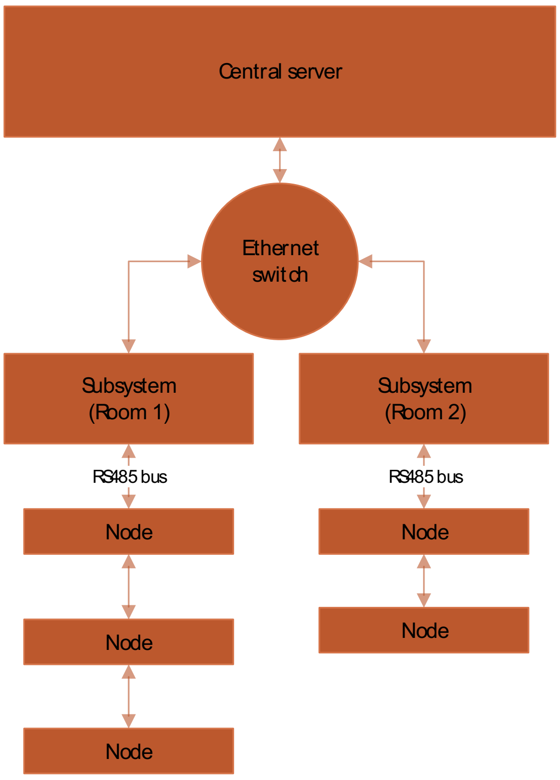

Figure 1: Block diagram of system architecture

Node must always fulfill certain properties and features which will be specified below in next section. Nodes contains various hardware peripherals, such as sensors, actors or controllers and they are connected to RS485 bus to communicate with the subsystem. Subsystems represent an extension of the existing architecture. Its default state is to delegate states and data between nodes and top layer, and the central server.

Central server replaces control unit by common computer on x86/64 platform with Linux open source operating system. This solution has low initial costs, and it is easily fixable when problem occurs. Server runs Java application with Spring Framework. It communicates with subsystems and with external services and mobile applications through REST API. It also contains web interface for configuration and monitoring. Central server receives node changes (button press, temperature change, etc.) and sets other nodes by user defined rules.

Subsystem keeps a local copy of user's defined rules to direct the connected nodes. If connection with the server is loss due to an error, nodes in the subsystem are still working. The local defined rules are evaluated by Java application called Controller, which shares information with the server application. The subsystem also contains C++ application called Handler. Its only purpose is to detect new nodes based on the demand of the Controller, and read or write data into nodes in infinite loop. The handle communicates with the Controller via JSON messages through TCP socket connection.

It is possible to use this architecture with minimal modification in other areas besides the smart home automation due to its modular architecture. For example, the subsystems can be places in hospital in every patient room. Individual nodes are placed in beds and detect patient's presence. Nodes can also work as emergency buttons to call nurses or to be placed on patient's body to measure data. The central server is located in the doctor's room and it notifies them about the changes by defined rules.

It is not necessary to use the whole architecture, as only nodes with different sensors and Handler application can be used. Those parts can be easily integrated to other scientific projects. The integration is fast and simple because of human readable JSON messages. There is only one requirement - to use programming language with TCP server connection and JSON parses support. Complete architecture is described in more detail in [12] [13].

II. Hardware and Software Implementation

Complete architecture is designed in a way that it does not require too much knowledge from hardware implementation area. It is possible to realize subsystem and nodes with very limited resources. Everything is designed to make available to experienced designers as well as to amateur constructers. For example, the popular Arduino platform can be used to implement nodes and the peripherals can be connected through breadboard so no soldering is required for testing implementations.

A. Subsystem

There are no specific requirements for subsystem's hardware. Every computer with UART interface and one GPIO pin can be used to develop the hardware. This pin is connected to data flow pins of RS485 transceiver. Bus is therefore, used in half duplex mode, where it is sufficient to use one pair of twisted cable. MAX485 from Maxim is used in testing implementation of nodes, in which popular ARMv7 architecture minicomputer Raspberry Pi 2 is used. It offers many processor performance and operating memory. Further, it is possible to use any other minicomputers with similar performance. UART interface is connected to MAX485. Maximum number of nodes in one subsystem is limited to 31 when using MAX485. This limit can be increased up to 127 nodes by using more advanced transmitter.

Only Java runtime and MySQL database is required for subsystem software. For compilation of Handler from code sources, C++ compiler with 2011 standard is required.

B. Nodes

Every node needs to have its type, channel count and address (identifier) specified. Node type means what kind of periphery (sensor, controller or actuator) is connected to node and which data are returned or accepted. Node type must have its identifier, access type, data count and data width in bits.

Node type can be, for example a switch. There is nothing to write into switch only to read its state and there are only two states: On or off. The two states can be represented with only one bit so the data width is equal to one. The data count are the same as there are no other information that the button can provide. The node has button state, which is available at any time and this state can be sent to subsystem immediately.

The next example is the color lightbulb. There is nothing to read from lightbulb, except it is only set based on color. However, it is possible to read and verify the latest state from the node. Color lightbulbs have three independent LEDs: Red, green and blue. The color lightbulb node type will have data width of 8 bits and data count of three. The values of data (from 0 to 255) are light intensity for LEDs.

Both examples are node types with immediate access. It means that data can be read or set constantly. In some node types, it is even necessary to read data constantly. For example, when the node type is button. If data from button node are not read fast enough, short button press could not be detected. To solve this, Handle must be able to read data from node with immediate read type as fast as possible.

The second group of node types are those with delayed access. Those nodes contain periphery which needs certain time frame to measure and process data. Delayed access types are therefore, read only. Those are the most often node types with some digital environmental sensors (temperature, humidity, air quality, etc.), which use some digital bus. If node would try to read data constantly, there would not be any or wrong data. Therefore, node with delayed access are waiting for measure command from subsystem. After some defined time to measure, data subsystem will read their data.

It is possible to split node types into four groups by their access type: Immediate read, immediate read and write, immediate write and delayed read.

Another property of node is channel count. Node type only specifies connected periphery but it is not counted as one of them. The number of nodes needed in house would be huge if the node could contain one periphery only. Therefore, node can have multiple identical peripherals and the channel count tells how many it is for example, eight channel switch node or two channel temperature sensor node.

The last property of node is its address. The subsystem must communicate with a particular node; therefore, nodes need to have a unique identifier, which is their address. The process of dynamic assigning address to node is described in further communication protocol introduction.

Handler does not have any information about which node types exist or which nodes are connected to it after the start up. This information must be sent from Controller via JSON message, and it is possible to define any new node type. The controller has already many node types built in his database. These are listed in Table 1, but the list has been shortened because there is currently more than 40 node types. ID means identifier of the node type, bits means the data length, count is the data count and access means type of data access.

From the hardware point of view, the nodes must contain microcontroller to be able to communicate with subsystem resp. Handler application. The chosen microcontroller must be connected to or to have integrated with RS485 transmitter. Microcontroller must have UART interface, if the transmitter is not integrated. Communication runs at 115200kbit per second speed and microcontroller must be able to work properly with this speed. It also must contain one significant analog input pin, which will be explained later in this paper. Other microcontroller's features like buses, timers or interrupts are dependent on the type of connected periphery.

Table 1
Various node types

ID	Description	Bits	Count	Access
1	Button	1	1	R
2	Switch	1	1	R
3	Light	1	1	RW
4	Dimmed light	8	1	RW
5	Dimmed RGB light	8	3	RW
6	Humidity sensor	8	1	DR
7	Temperature sensor	8	2	DR
8	Dimmer	8	1	R
9	Door lock	3	1	R
10	Regulated fan	16	1	RW

Persistence memory like EEPROM or others must be presented in the microcontroller for protocol implementation. Only two bytes of persistence memory id needed, approximately 64 bytes of RAM memory and 1 kilobyte of program memory were used. These numbers are indicative and depend on the used microcontroller, programming language or efficiency of language compiler.

Schematic diagram of node with precise digital temperature sensor DS18B20 is shown in Figure 2. This diagram is intended only for testing purposes in laboratory environment. Node type is temperature sensor (type id 7) with delayed access and two eight bit data which are returned from node.

The diagram uses very simple and cheap 8-bit microcontroller PIC16F1813 from Microchip Company. Circuit requires stabilized 5 volt input through JP2 pin header. JP2 also contains doubled A and B pins for RS485 bus connection. Pin header JP1 is for terminating resistor connecting and this header must be connected physically on the last node on the bus.

It is possible to design very small printed circuit board because schematic contains small amount of parts, even while using classic, non SMD parts and when leaving enough space for manual assembly. The final size is only 22 to 22 millimeters, if the two layered routes are used as seen in Figure 3. If SMD parts are used, the final printed circuit board size is even smaller.

III. Protocol Description

Subsystem is the master and controls communication flow, while nodes are the slaves that wait for the command because RS485 is used in half duplex mode. The whole protocol is therefore in request-response style. The subsystem sends request to every node or one specific node and waits for response. The whole protocol is binary and there is no fixed message length. Every message needs to have data as shown in Table 2.

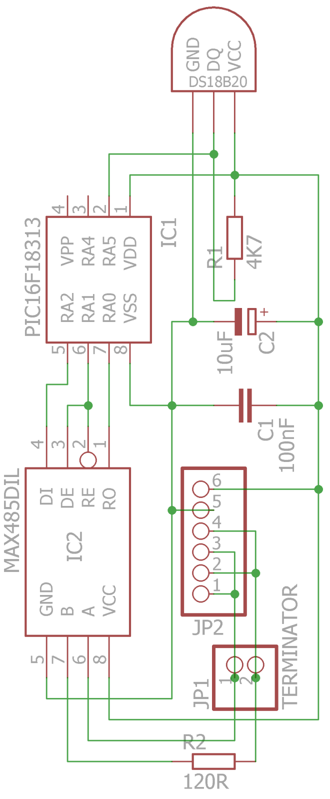

Figure 2: Schematic of temperature node

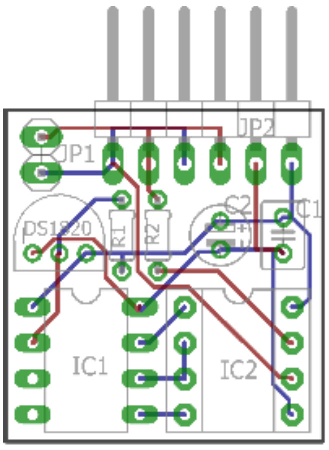

Figure 3: Printed circuit board of temperature node

Table 2
Description of message format

Byte Number	Description
1	Start character
2	Message length
N	Message data
N+1	CRC8 checksum
N+2	Ending character

The first byte of message is the start character. Its binary value is 35 which corresponds with ASCII character #. The second byte contains information about its own data length. The minimum own data length is one byte and the maximum allowable length is 32 bytes. The next bytes are the message data itself. The first message byte must be a message identifier which specifies the message content and meaning. A list of identifiers is shown in Table 3. Some messages do not contain any other data because they are only commands for nodes to do some actions. The checksum follows after its own message data. It is calculated from the message length byte and own message bytes and is calculated by exclusive disjunction with pre calculated values of x^8+ x^5+ x^4+1 polynome. The last byte is the ending character with binary value 36 which corresponds with ASCII character $.

Table 3
List of protocol messages

MESSAGE NAME	ID
MESSAGE_DISCOVERY	0
MESSAGE_DISCOVERY_RESPONSE	1
MESSAGE_DISCOVERY_ADDRESS	2
MESSAGE_DISCOVERY_CHECK	3
MESSAGE_DISCOVERY_CHECK_OK	4
MESSAGE_REQUEST	10
MESSAGE_GET	11
MESSAGE_SET	12
MESSAGE_MEASURE	13
MESSAGE_PING	14
MESSAGE_PING_RESPONSE	15
MESSAGE_RESET	16

A complete protocol contains only twelve different messages as seen in Table 3. They can be divided into two logical groups. Messages for address are assigned with identifiers 0 to 4 and for regular communication with identifiers 10 to 16. Identifiers from 5 to 9 are reserved for future improvements.

A. Assigning address

The process of assigning address must be started by user or other application on the central server. These messages do not need to be implemented to testing nodes. They can have their address hard wired into firmware. It is important to pass the list of existing nodes (their channel count and address) and the node types in Handler. Further, the uniqueness of addresses must be guaranteed; otherwise, the system will not work correctly.

The following messages are ordered in accordance to the process flows in the real scenario.

a. MESSAGE_DISCOVERY

Handler starts addressing the process with this message. It does not contain any other data because it is only a command that not to address the nodes to send their information. Nodes with already assigned address ignore this message as well as any other addressing messages.

b. MESSAGE_DISCOVERY_RESPONSE

This message is a response from nodes on request from MESSAGE_DISCOVERY. Ungrounded analog pin is used in this part. If two or more new not addressed nodes are connected to subsystem, there is no way to identify which one of them should respond first, second and etc. They would began sending responses at the same time immediately after receiving MESSAGE_DISCOVERY.

Therefore nodes take electric noise from environment, which is inducted on analog input pin. They measure this value twice and the results are multiplied between themselves and used as variable X. Now, they wait for X milliseconds before sending their response. This response contains this pseudo random number, channel count and node type. Collision may occur when addressing large number of nodes at once. Central server assigns addresses to nodes which did not collide. Notification about collision is send to user or other application. Handler does not send MESSAGE_DISCOVERY itself again. It is necessary to receive JSON message with command from Controller or other used application.

Real data for four channel switch node are shown in Figure 4. Own data length is five bytes. It contains message identified, lower and higher byte of random number, node type and channel count.

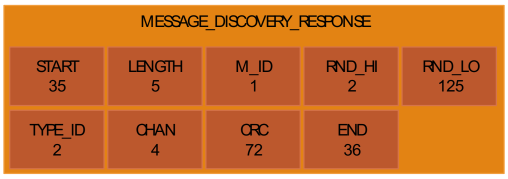

Figure 4: Message discovery response content

c. MESSAGE_DISCOVERY_ADDRESS

Handler sends list of new nodes to Controller when receives MESSAGE_DISCOVERY_RESPONSE from new nodes. Pseudo random number is sent because this new nodes do not have addresses. Higher layer then assigns new address and pairs it with pseudo random number. Therefore it is possible for nodes to identify if new address is available for them. Handler sends MESSAGE_DISCOVERY_ADDRESS to nodes with new address and pseudo random number.

d. MESSAGE_DISCOVERY_CHECK

Node saves its new address only into RAM memory when MESSAGE_DISCOVERY_ADDRESS is received. Node sends back to subsystem MESSAGE_DISCOVERY_CHECK with its new address and pseudo random number for checking.

e. MESSAGE_DISCOVERY_CHECK_OK

This message means that node has correct address when data from MESSAGE_DISCOVERY_CHECK are correct. Node is ready for switching into its normal function. Handler therefore sends MESSAGE_DISCOVERY_CHECK_OK only with new address of node. Node saves address into EEPROM memory or any other available persistent memory after receiving this message.

B. Getting the data

These messages are being used when the system is running correctly and reads or writes data. Handler application is running in infinite loop and is getting data from nodes with immediate read access.

a. MESSAGE_REQUEST

This message is sent to nodes with read access. Address of node is part of this message to request specific node. Response with data is sent if node successfully verifies checksum and address in message and is equal to its own address. Other nodes with different address ignore this request.

b. MESSAGE_GET

Nodes are responding with this message after receiving MESSAGE_REQUEST. Data and node address are included. Node sends data for all its channels at ones. Data are sent bit by bit according to data count and data width because protocol is binary, for example node with three channels and with one variable (data count is one) with three bit width. Aligning of bits in data byte is shown in Table 4.

Transmission of data is memory efficient due to bit aligning. Three channel node with three bit width would allocate 3 bytes if data would not be aligned bit by bit. Thus, only one whole byte and one bit from other is allocated.

Table 4
Example of node's data

Bit number	Byte 1	Byte 2
Bit 1	Ch. 1, bit 1	Ch. 3, bit 3
Bit 2	Ch. 1, bit 2	Not used
Bit 3	Ch. 1, bit 3	Not used
Bit 4	Ch. 2, bit 1	Not used
Bit 5	Ch. 2, bit 2	Not used
Bit 6	Ch. 2, bit 3	Not used
Bit 7	Ch. 3, bit 1	Not used
Bit 8	Ch. 3, bit 2	Not used

c. MESSAGE_SET

Subsystem sets data of nodes with write access. There are node data and address of node contained in message. Data are aligned similarly as in MESSAGE_GET message.

d. MESSAGE_MEASURE

Subsystem sends this message as command to measure new data because nodes with delayed reading need time to acquire peripheral data. There can be situation when nodes are incapable to send their data if these nodes are measuring data of peripheral right now. This message eliminates this issue. Node has plenty of time to measure data after this message is received. This data is stored in RAM memory and after receiving MESSAGE_REQUEST message nodes responds with MESSAGE_GET same as when instant read nodes are called. Delay between MEASSAGE_MEASURE and MESSAGE_REQUEST is set to ten second at least in Handler. This time is sufficient to any type of commercially available sensor to read data.

e. MESSAGE_PING

There is no way to determine if nodes for write only access are working correctly because they are not returning any data. MESSAGE_PING is designed to test those nodes functionality. This message contains node address only.

f. MESSAGE_PING_RESPONSE

When node receives MESSAGE_PING message it will respond with MESSAGE_PING_RESPONSE message. Node address is part of this message.

g. MESSAGE_RESET

The address of node is removed from permanent memory when receiving a MESSAGE_RESET message if there is a need to remove it for testing purposes.

IV. Conclusions

Designed and implemented system for smart home automation expands traditional two layer architecture of control unit and modules. This article presents low level communication protocol and hardware solution.

A new three-layer architecture replaces the central server instead of the control unit on the top layer. It contains user defined rules, which can be changed anytime when the user demands it. Subsystem is the middle layer and represents significant improvement over the traditional architecture. During any system malfunction as such, functionality in rooms where the present subsystem is kept. Low layer called nodes represents end points. Every node needs to fulfill defined properties and parameters. Due to this standardization, it is possible to have a simple communication protocol to get and set node states and assign unique address.

Communication protocol and architecture represent time and cost available solution on how to integrate existing projects that contains data from real sensors or peripherals. Those can be any peripherals, which are available on the market and can be attached to microcontroller. This microcontroller must be capable of running presented protocol and operating selected peripheral. It is possible to use only selected parts of the system thanks to the modular architecture, for example, to use only nodes and Handler application that can provide sensor readings for external application such as neuron networks [14] or fuzzy logic.

Acknowledgement

The support of Czech Science Foundation GA_CR #15 11724S is gratefully acknowledged.

References

[1] Lee J., Su Y. and Shen C. , 2007. A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi, *IECON 2007 - 33rd Annual Conference of the IEEE Industrial*

[2] Inels.com, Smart Wiring, Smart Home - iNELS.com, 2015. [Online]. Available: http://www.inels.com/. [Accessed: 12- Oct- 2015].

[3] Loxone.com, 2015. The Loxone Smart Home - Your Smart Home System Loxone, [Online]. Available: http://www.loxone.com/enen/smart-home/overview.html. [Accessed: 28- Oct- 2015].

[4] Fibaro.com, 2015. Fibaro - Z-Wave smart home solution, [Online]. Available: http://www.fibaro.com/. [Accessed: 01- Oct- 2015].

[5] Foroughi H., Aski B. and Pourreza H. 2008., Intelligent video surveillance for monitoring fall detection of elderly in home environments, *2008 11th International Conference on Computer and Information Technology*

[6] Mubashir M., Shao L. and Seed L., 2013. A survey on fall detection: Principles and approaches, Neurocomputing, 100:144-152.

[7] Rashidi P. and Cook D., 2009.Keeping the Resident in the Loop: Adapting the Smart Home to the User, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 39(5):949-959

[8] Fadlullah Z., Fouda M., Kato N., Takeuchi A., Iwasaki N. and Nozaki Y., 2011. Toward intelligent machine-to-machine communications in smart grid, *IEEE Commun. Mag.*, 49(4):60-65,

[9] Jakes M., Brozek J., 2015. Connection of microcontroller and microcomputer to distributed simulation, 27th European Modeling and Simulation Symposium, EMSS 2015. Bergeggi: Rende, France, 282-288

[10] Brozek J., Jakes M., 2015. Hardware libraries for online control of interactive simulations, *27th European Modeling and Simulation Symposium,* EMSS Bergeggi: Rende, France, 295-300.

[11] Krejcar, O., Spicka, I., & Frischer, R. 2011. Implementation of full-featured PID regulator in microcontrollers. *Elektronika ir Elektrotechnika,* 113(7), 77-82.

[12] Stepan J., 2015. Design and implementation of smart device's hardware and software for communication in smart home', Bachelor, University of Hradec Kralove,

[13] Horalek J., Matyska J., Stepan J., Vancl M., Cimler R. and Sobeslav V., 2015. Lower Layers of a Cloud Driven Smart Home System, New Trends in Intelligent Information and Database Systems, 219-228.

[14] Mikulecky, P., Cimler R., and Olsevicova K. 2012. Outdoor Large-scale Ambient Intelligence. 2012 *Proc. 19th IBIMA Conference*. Barcelona, IBIMA