Development of a Domotic Module for DomoBus

António B. Sousa
DEEC, Instituto Superior Técnico

***Abstract*—DomoBus is a home automation system being developed in an academic context. It aims to be a solution for the problems presented by current systems, such as the lack of support for interoperability between different technologies. DomoBus proposes a generic model for a domotic device, which is characterized by a set of properties, and a set of well-defined messages to interact with those properties. A Control Module (CM), in DomoBus, is a hardware component that may control multiple domotic devices of different types. The main objective of this work is to design and implement a software architecture for CMs that will follow the DomoBus approach and simplify the future development of applications that will interface with different sensors and actuators. The designed software will be executed in a cooperative multitasking scheduling that includes time management and an implementation of the communication protocol. It will also be offered a model for the development of DomoBus applications in which they access the properties through a simple API, as if they were local variables. In DomoBus, a Gateway Module (GM) is a hardware component that interfaces the CMs with a computer. The software that supports the desired functionalities will also be developed. The software developed for the CMs and GMs is intended to be implemented in devices like Arduino boards, that present limited hardware resources. For the validation of the presented concepts, a prototype with two CMs and a GM connected to a computer was developed. The obtained results satisfy the established objectives.**

***Index Terms*—home automation, domotic systems, KNX, LonWorks, DomoBus, domotic module.**

I. Introduction

THE creation, development and application of home automation systems are processes related to the Domotics field of study. The first kind of this type of systems appeared in France in the 1970's [1]. These were centralized systems, in which a central control unit was responsible for the management of the whole system. By using various units and applying a hierarchy around them, the first distributed systems appeared. These systems evolved into integrated systems in the beginning of the 1980's.

Nowadays, one can easily verify that there are plenty of home automation systems available in the market, like KNX and LonWorks, that already provide a substantial amount of functionalities. However, all these systems present the problem of being incompatible with one another [2], as they use specific circuits and devices. These technologies are hard to install, to configure and to adapt the user needs and preferences. Besides, the software responsible for their management is proprietary.

In this context, it is easy to understand that one should invest in a system that could present a solution to the enumerated problems and provide the same or better features than the ones offered by existing systems. That is precisely the main goal of DomoBus, an open-source home automation system created in an academic environment for teaching purposes. In DomoBus, devices are generic entities characterized by a set of properties. A lamp, for example, may be characterized by two properties, namely its On/Off status and the level of intensity of the light. The interaction with the devices is made through a simple set of messages: GET (to get the value of a device's property), SET (to modify a property's value and, consequently, change the state of the device) and NOTIFY (transmitted automatically by the devices when one of its properties has changed its value). The DomoBus architecture is divided into two interconnected levels. The first one, called Supervision Level (SL), is responsible for the global management of the system, the high-level interface with the users and control of the system behaviour. The other one, designated Control Level (CL), is constituted by Control Modules (CM). A CM is a hardware component similar to an Arduino board which is responsible to interface with sensors and actuators. A CM is able to control multiple domotic devices of different types. These modules are interconnected through a network, called DomoBus Control Network (DCN).

The objective of this work is the development of a modular software architecture for a CM that will support all the communication requirements and the management of devices' properties. The main goal is to offer a simple and generic API for device developers that will ease the integration in DomoBus of new devices. Developers just need to interface with the devices' hardware and map their relevant characteristics into properties, which will be managed through the given API. The software architecture of a CM will be composed of three main layers: NET, MESSAGE and PROPERTIES. All these layers, in conjunction with the control applications, will be executed in a cooperative multitasking scheduling.

The NET layer is responsible for the transmission and reception of packets, allowing each CM to interact with other CM and with the applications of the SL. The MESSAGE layer processes the DomoBus messages, received through the NET layer. The PROPERTIES layer will offer a simple API to the developer of a domotic device that allows an easy access and manipulation of its properties. Developers just need to interface with the devices' hardware and map their relevant characteristics into properties. The given API will allow the automatic update of a property's value and, through the MESSAGE layer, transmit the corresponding NOTIFY message to the SL.

Given the involved complexities, it will also be developed an application that handles the configuration of a Control Module, according to a set of applications, devices and properties defined by the user. This application will also generate a basic

structure for the code of the CM applications, exposing the developed API and simplifying the access to the property values.

With the objective of facilitating the integration of the developed CMs in the DomoBus system, a modular software architecture for a Gateway Module (GM) will also be developed. A GM is a hardware component similar to a CM that allows the connection of the Control Modules to the system's Supervision Level.

The remaining of this document is organized as follows. Section II provides an overview on the development of control modules for the existing home automation systems and introduces the DomoBus system. Section III describes the software architectures and hardware requirements proposed for the Control Modules and Gateway Modules. Section IV includes a description of the processes implemented in the modules' software and presents the hardware devices used in the implementation. Section V presents the tests performed on a simple prototype. Section VI presents the conclusions of this work.

II. Related Work

Home automation systems are based on networks of sensors and actuators. Sensors gather all kinds of data concerning the state of a house as such luminosity or temperature. With the gathered data, the system performs actions in the installed set of actuators according to what it is defined in its control software.

In the assembly process of these kind of networks, there are two main approaches to take into consideration [3] [4]:

- **conventional/centralized control systems**
- **control networks** or **bus technologies**

The second approach brings more advantages, especially if one desires a system that controls a substantial number of devices. Given the decentralized structure of the network, sensors and actuators may communicate with one another without having to exchange information with a central control unit. Besides, the installation and configuration of this kind of networks is a more simple and flexible process [3].

Given their simplicity and flexibility, decentralized systems represent the present and the future of home automation systems. As such, it is important to analyse some of the main systems of this kind, like KNX and LonWorks, in order to gain the necessary knowledge on the subject.

A. KNX

KNX is the most popular home automation system in Europe. In this system the different modules can be separated into two separate categories [3]:

- System Devices, such as power supplies or the programming interfaces;
- End Devices or Bus Devices (BD), associated with the different sensors and actuators controlled by the system. It is important to notice that each one of these modules was developed for the control of a specific type of device. This means one cannot have a lamp and a light sensor controlled by the same BD, for example.

Taking into account the main objectives of this work, it is important to perform an analysis of the structure of the Bus Devices. KNX Bus Devices are constituted by two components, a Bus Control Unit (BCU) and the application module. The BCU is responsible for controlling the access of the device to the communication bus and for the management the actual end device (that sometimes comes with its own microcontroller, that replaces the BCU in this task). This is a device with many possible designs that, fundamentally, is constituted by two sub-components: a Transmission Module (TM) and a Controller [3]. The TM is responsible for implementing the communication protocol by which the BD exchanges information.

Concerning the Controller, this sub-component is basically an NEC, ATMega or Texas Instruments generic 8-bit microcontroller [3]. Regarding the programming of a KNX system, there are two possible configuration modes to take into consideration [3]:

- Easy Mode (E-Mode), in which no software is needed;
- System Mode (S-Mode), in which a computer application designated ETS is used to configure the system. This is the most common configuration process. The major flaw of ETS is the fact that this is a proprietary software, which means that one can only get little or no information regarding the development of the system's control modules, like programming methodologies or software libraries.

B. LonWorks

The LonWorks platform, developed by Echelon Corporation, is a very complex and sophisticated system that is used in various industries and normally installed in office buildings, malls or hospitals [5].

In a LonWorks Control Network, the different control devices communicate using a common protocol, the ISO/IEC 14908-1 Control Network Protocol (CNP). In CNP, there are many types of communication transceivers available that support both the transmission and reception of data between devices. These support communication over twisted pair, link power, power line, radio-frequency, fiber optic, and infrared media. Each device implements the communication protocol and performs control actions. These control devices may include communication transceivers to allow the coupling of the device with the communication medium.

In order to achieve economical and standardized deployment Echelon designed the Neuron Core, that includes multiple processors, memory and interfaces for communication and I/O [4]. Echelon also developed a complete operative system for the Neuron Core called Neuron Firmware, that includes a complete implementation of the CNP. The Neuron Core is available as a standalone component called the Neuron Chip. There are a few different types Neuron Chips available in the market, with their own interfaces, speeds and memories. In order to reduce costs, these cores may be combined with communication transceivers. The resultant devices are called Smart Transceivers.

The LonWorks applications are programmed in a special language called Neuron C [6]. This language is based on the

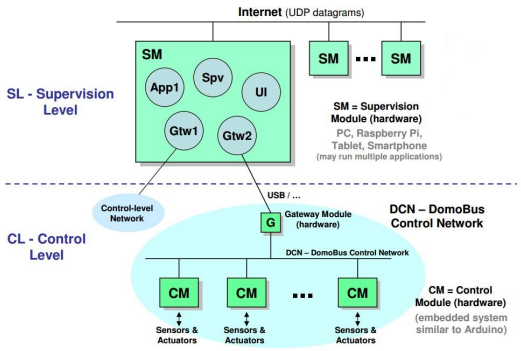

Fig. 1. DomoBus system architecture

classic ANSI C and it was specially conceived for the LonWorks Neuron Chip and Smart Transceivers. This language includes network communication, I/O and event-handling extensions for the ANSI C.

C. DomoBus

DomoBus is a home automation system created in an academic environment for teaching purposes that includes features like inter-protocol compatibility and ease-of-use by common users and programmers. Figure 1 illustrates the architecture of the DomoBus system.

It is possible to verify that the system may be divided into two different levels. The Supervision Level, constituted by the Supervision Modules (SM), is responsible for the management of the system. In this layer, it is possible to implement some sophisticated automatisms concerning all the sensors and actuators controlled by the system. The Control Level (CL), constituted by Control-Level Networks, is responsible for the interface between the Supervision Level and the various control points (sensors and actuators). These Control-Level networks may belong to different technologies, like KNX or LonWorks, or may be related to DomoBus itself. In the latter case, such networks are designated DomoBus Control Networks (DCN). A DCN is constituted by various Control Modules, responsible for the control of the different home devices, and by a Gateway Module (GM) that provides a connection between the SMs and the CMs.

The Control Modules are hardware devices, based on generic boards such as Arduino, that interact directly with the home devices that one wishes to control with the DomoBus system. A CM to is able run different applications and control multiple devices simultaneously. This is a specially important feature in homes where the number of control points is quite substantial. By taking this approach, it's possible to reduce costs and keep a simple system architecture.

In DomoBus the devices to control are considered to be generic entities characterized by a set of controllable properties. If one wishes to control a common lamp, for example, two properties may be taken into consideration: its On/Off status and its light intensity. DomoBus will interact with the different home devices by controlling its properties. This interaction is based on a simple communication protocol, with three main types messages: GET, SET and NOTIFY.

The GET message is sent in order to get a property value of a certain device. The respective reply message sent by the device, with the requested value, is called A_GET (Answer to GET). By using the SET message, it is possible to change a property value of a certain device. The NOTIFY message is sent by the CMs to the SMs when the properties of their associated devices change their value.

III. Proposed Solution

A. Requirements

DomoBus is a system conceived to support the automation of Super Automated Houses (SAH), which are characterized by their high number of control points and, consequentially, a considerable amount of Control Modules. The modules to use should be based on the most simple and economical microprocessors possible, balancing other factors such as power consumption, accessibility and support. As such, it is important, first of all, to establish the minimal hardware requirements for the microprocessors of the modules where the developed software is going to be implemented.

Taking into account the characteristics desired for the microprocessors, one should choose devices based on 8-bit microprocessors (MCU) in spite of 16-bit or 32-bit microprocessors. It is possible that 16-bit or 32-bit MCUs would present better features and a better performance. However, 8-bit microprocessors are enough for the implementation of the developed software and, in a general way, are simpler cheaper than the others. It is also important to notice that the software developed for 8-bit microprocessors can also be implemented in 16-bit or 32-bit microprocessors.

Regarding flash memory, 16 KB memories are expected to be enough for the developed software. Concerning RAM memory, some initial estimations were performed given the number and type of the data structures that will support the desired functionalities. 512 bytes probably would be enough, but to give a safety margin it was decided that the microprocessor should have 1 KB or more of RAM memory. Regarding EEPROM memory, it was estimated that 1 KB is sufficient for the support of the functionalities where this kind of memory is needed.

The microprocessors of the modules to use should also include the following components:

- interface peripherals, such as Serial Peropheral Interface (SPI), I^2C and UART, to support serial communication and interface with different communication modules;
- digital I/O pins (no less than 16), and some should support PWM generation;
- analog I/O ports, with at least 2 pins (with their respective Analog-to-Digital Converter (ADC) for the control of analog peripherals;
- 2 timers at least (one to be used by the system and other being available for applications);

It is possible that some of the mentioned components won't be needed in a certain context, like some of the interface peripherals or the ADC, for example, but their availability increases the overall flexibility of the system without a significant cost impact.

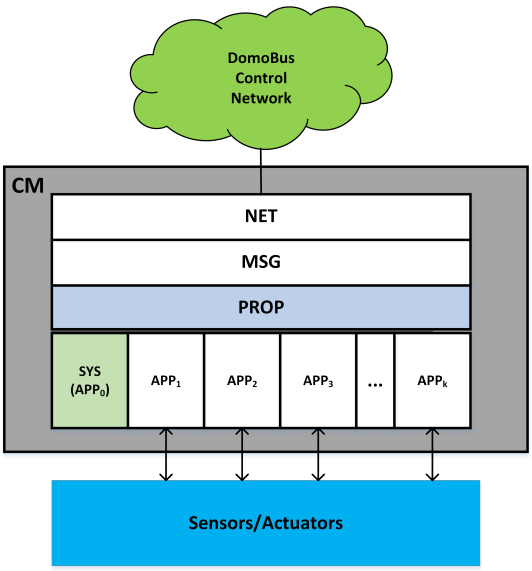

Fig. 2. CM software architecture

It's also recommended that the chosen control devices are easy to acquire (particularly if one needs to swap devices as a consequence of a malfunction, for example) and also that the process of developing software for these devices is simple and well-supported. These features may be found within the Arduino boards scenario, making them strong candidates for being used as a CM or a GM in the context of a DCN.

B. CM Software Architecture

Figure 2 presents the proposed software architecture for DomoBus Control Modules.

The software architecture can be divided into 4 separate layers: NET, MSG (MESSAGE), PROP (PROPERTIES) and the applications.

The NET layer corresponds to the software module directly responsible for the exchange of data packets within the DCN. There are two types of messages to take into consideration: Commands (messages that specify an action to be perform at the destination node) and Answers/ACK (answers or acknowledgements of previous Commands);

The DCN can be implemented using different media, such as twisted pair cables, RF or even the house's power supply network. Depending on the type of communication media, different transceivers and communication modules can be used. The NET software module will be responsible to interface with the used communication hardware.

The CM includes one single communication buffer to store a message received or a message to be sent. It was decided to not use more buffers to support the communication process given that the developed software will be implemented in devices with very limited hardware features. The addition of more buffers would reduce the available memory, not to mention that the management of that set of buffers would be a more complex process that wouldn't be rewarding because. This buffer is managed by the NET and MSG layers, with recourse to some flags that give information regarding the state of the buffer.

Most of the time, the node is waiting for a message to be received, switching out of this state to transmit Commands or Answers. The messages to transmit are generated in a short period of time, without causing any kind of impact in the node.

The MSG layer corresponds to the CMs' software section that constitutes an interface between the messages exchanged with the DCN and the properties associated with the devices controlled by the Control Modules. In this layer, there are two main functionalities to be taken into consideration. The first one consists in the treatment of the received messages and the assembly of the respective replies. This is a process that begins with integrity tests to validate the received data. More specifically, it is verified whether the node, application, property and device addressed in a received message are valid or not. If an error is found, an Answer is sent back to the origin nodes with an appropriate error code. On the contrary, is no issues are found in a received message, the next phase of the message treatment may proceed. This phase depends on the type of message received:

- if the received message is an Answer to a previously sent Command, this will be registered in the property management data structures. If by any chance the received reply isn't associated with any of the controlled properties, the message is simply discarded;
- in case the message is a Command, an action in the addressed property/device pair is performed according to the received message's type. If a GET message is received, a reply (called A_GET) is sent with the desired value. In case the received message was a SET, the property value is updated, and a reply is sent back signalling the success of the operation. The reception and treatment of NOTIFY Commands aren't supported by the Control Modules.

The second functionality of the MSG layer consists on the assembly of the NOTIFY messages that are sent when a property value has changed. Although this is the typical procedure, it is also possible for a CM to interact directly with other CM, sending SET messages and, therefore, being able to actuate in their devices.

The PROP (Properties) layer corresponds to the software section responsible for the management of the properties of the various devices controlled by the Control Modules. In comparison with the other software modules, the PROP layer presents some substantial differences as it is simply constituted by a set of data structures and functions to interface those structures with the MSG layer and the control applications. There are three types of properties handled by DomoBus:

- 8_BIT: the most typical type. An 8_BIT property value is composed by a single byte of data;
- 16_BIT: property value is composed by two bytes of data;
- D_ARRAY: this type's name stands for "DomoBus_Array". It consists of a byte array, and it is generally used when the property's absolute value can't be represented in the other forms. The first byte of a D_ARRAY property indicates the number of data

bytes of the property value.

All the properties of a CM are stored in specific arrays (one for each type of property). These arrays are accessed by the MSG layer and by the control applications. The access to the properties is managed with a pair of control flags designated events, triggered every time a new value is assigned to a certain property. There are two types of events to be considered: RX-events, used to signal that a property was changed by a SET Command, and TX-events, which are associated with the changes in the properties' value imposed by the control applications.

The applications (APPS) are the software modules that are actually responsible for the control of the various sensors and actuators connected to the Control Module. The control of the various devices is based on the values of their properties. All the CMs include a special application designated SYS, which is responsible for the management and supervision of the Control Module itself. The only device controlled by this application is a LED which is inherent to the CM, characterized by one property which is its number of blinks. This property may be used as a debug tool that gives information regarding the state of the CM at a certain moment in time. In addition to the referred property, the SYS application controls a special type of properties called Application Parameters (AP). The AP can be seen as the input specifications of the remaining control applications.

To ensure a good modularity of the software and other mechanisms for multitasking, it was decided to incorporate in the software architecture a basic operating nucleus. The operation of the different tasks and control applications can be divided into two distinct phases called INIT and EXEC. In the INIT phase, that occurs only once, all the tasks and applications execute sequentially a special function called init() that includes initialization actions that setup the Control Module, the connected hardware devices (both the devices to control and the external communication modules) and the data structures for the execution cycle EXEC. The EXEC operation stage occurs uninterruptedly until the deactivation or the reset of the CM. In this phase, all the tasks and applications execute another special function called task(), that implements the logic related to their role within the overall software architecture. Given the fact that there is only one single processor available, it was decided that the various task() functions are executed in this processor according to a non-preemptive scheduling in a "round-robin" fashion. The cooperative scheduler assigns the processor to the various task() functions sequentially and periodically. It is intended that any execution "lap" doesn't last more than 1 ms.

In addition to the already mentioned tasks and applications, it is important to mention two more tasks: TIME and EEP. The TIME task provides timing functions to the other tasks and applications, offering a multiplicity of (software) timers with different periods. The EEP task, in turn, features functions for the management of the Control Module's EEPROM memory, which is a global resource, but can be used by different tasks. In particular, the EEP task manages writing in the EEPROM, operation that can take a long time to execute.

C. DCN Gateway

In order to implement a fully operational DCN, there was a need to develop a software architecture for Gateway Modules (GM).

The GMs hardware requirements are similar to the CM's, with a few differences. A GM doesn't control any kind of external peripherals except the hardware communication module used in the exchange of packets with the remaining nodes of the DCN.

The software architecture developed for the GMs is composed by two main modules: NET and SM_COMM. The NET layer has exactly the same function of the namesake layer in the Control Module's software architecture. The SM_COMM layer, in turn, is responsible for the exchange of data between the GM and the SM's Gateway application. This layer has to be adapted to the protocol used for the communication with the SM.

The SM_COMM layer and the NET layer share two communication buffers where the various messages exchanged in the Gateway Module are stored. One buffer is assigned for the storage of the messages received from the DCN which are sent to the SM, and the other one is used to save the messages received from the SM that need to be sent to the DCN. The contents of the buffers are controlled via some flags that give information regarding their state.

IV. Implementation

The software developed for the CMs and GMs of a DCN was implemented in Arduino boards, as proposed. Hardware-wise, Arduino boards not only present the necessary components but also simplify several aspects of the software development process. In fact, the Arduino platform provides all the tools and libraries which are necessary for the programming of the microprocessor incorporated in the boards and, by extension, of its components. Taking into account all the boards available in the Arduino platform, it was decided to implement the software architectures for the CMs and GMs in Arduino UNO boards, for being the most simple boards of all the product line.

A. ATmega328P

The Atmel ATmega328P microprocessor is the 8-bit microcontroller used in the Arduino platform boards that are based on 8-bit microprocessors, namely Arduino UNO, Nano and Mini. This MCU includes the following features [7]:

- 32 KB of flash program memory, with 10000 write/erase cycles;
- 2 KB SRAM;
- 1 KB of EEPROM, with 100000 write/erase cycles;
- Two Master/Slave SPI Serial Interface;
- programmable Serial USART;
- Byte-oriented 2-wire Serial Interface, compatible with Philips I^2C
- 23 programmable I/O lines;
- 6 PWM channels;
- Two 8-bit timers;

- One 16-bit timer;

In this work, the ATmega328P microprocessor was programmed in pure C language. In this manner it's possible to divide the code into several code and header files, giving modularity to the program, and one has a full control of the code that is loaded into the flash memory (avoiding the inclusion of unnecessary data related to other boards) and of the microprocessor's control registers.

B. nRF24L01+ Communication Module

Regarding the communication within the network, it was decided that the communication between devices should be performed in a wireless fashion through nRF24L01+ radio modules. These are cheap and small radio modules which are fully compatible with Arduino boards, simple to use and present low energy consumption rates. These modules also provide a set of helpful communication features like automatic packet validation and retransmission. The nRF24L01+ radio modules' main features are the following [8]:

- **RF Data Transmission Rates** - 1 Mbps or 2 Mbps;
- **Operation Frequencies** - 2,4 GHz ISM band.
- **Non-Overlapping RF Channels** - 126 channels when operating at 1Mbps and 63 when operating at 2Mbps;
- **Transmission Power** - the output power goes from -18 dBm to 0 dBm (the power verified in Bluetooth standard radios);
- **Host Interface** - 4-pin hardware SPI, that supports a maximum data transmission rate of 8 Mbps. Three separate TX and RX FIFOs, with a capacity of 32 bytes, and 5V tolerant inputs are also included.

The control of the nRF24L01+ modules is based on a set of commands that allow not only the transfer of the data to the DCN but also the handling of the modules' control registers.

C. NET Task

In the INIT pahse, the NET task initializes and configures the microprocessor's SPI and the nRF24L01+ radio module, preparing the devices for the communication operations. In the EXEC phase, only one message reception or transmission is performed, depending on the activated buffer control flag associated with state of the communication buffer shared between the NET and MSG tasks. There is a total of five states to take into consideration: IDLE, BUSY, TX, RX, and REPLY. When the buffer is in the IDLE state, any kind of message may be loaded. The buffer is in the BUSY state when a message is being loaded into it. In the TX state, the buffer contains a Command to be transmitted. When the buffer contains a received message, the RX flag is activated. The REPLY flag, in turn, is activated when the buffer contains an Answer message to be transmitted.

Whenever the NET and MSG tasks try to access the buffer, they verify whether the buffer is available or not. This availability depends on the activated control flag and also on the type of data that it is intended to load into the buffer.

- if the BUSY flag is activated, the NET and MSG tasks must wait until the loading process gets finished;
- if the TX flag is activated, the buffer may be overwritten in order to store messages received in the NET layer or Answer/ACK messages to transmit;
- when the RX flag is triggered, only an Answer/ACK message to transmit may be loaded into the buffer.
- if the REPLY flag is activated, no other messages may be placed into the buffer.

These rules were established taking into account that the reception of new data in the NET layer has a priority over the transmission of messages, with the exception of Answer/ACK messages.

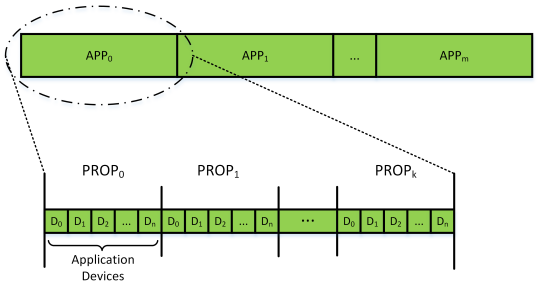

Fig. 3. Basic structure of property data arrays

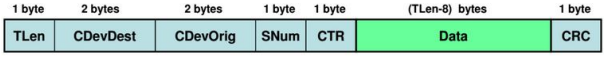

Fig. 4. DomoBus Control Level packet

D. PROP Layer

The property data, stored in arrays, is arranged according to the structure depicted in Figure 3. Three data arrays were allocated given the three possible property types.

E. MSG Task

The MSG task corresponds to the software section in which occurs a direct interaction between the DomoBus messages exchanged in the NET task and the properties of the controlled devices contained in the PROP layer's data structures. The DomoBus messages have the structure illustrated in Figure 4. The first byte, called TLen, contains the number of bytes of the packet. The fields CDevDest and CDevOrig are address fields that identify the node, application and device the packet was sent/received to/from. The SNum field contains the sequence number of the packet. The CTR field specifies the message type and the operation to perform (Command/Answer, GET/SET/NOTIFY). The Data field contains information regarding the type of the property, its ID and value. Finally, the last byte is a check byte used to ensure the integrity of the message in the communication process.

One of the main functionalities of the MSG task is the treatment of the DomoBus Control Level Data Frames received in the NET layer. The first step of this process consists in the computation of the index of the property to which the received message is destined. This computation is based on the destination values contained in the received packet, namely the

application number, device number, property type and property ID.

The next step depends on whether the received message is a Command or an Answer/ACK, a fact indicated by the flag ACK in the CTR field of the data frame.

In case the received message is an Answer/ACK, its treatment process gets finished after disabling the TX-event that generated the respective Command. If the received message is a Command, the treatment process proceeds with the execution of the action indicated by the Opcode of the CTR field of the data frame. If one receives a GET Command, the destination property's value is retrieved from the respective data structure. On the contrary, in case one receives a SET Command, the value contained in the frame's data field is loaded into the property values' data array, and the respective RX-event is triggered. After these operations, the respective Answer/ACK message is built in the buffer and treatment process gets terminated.

Regarding the events, which are the control flags used to signal modifications in properties' values, these were implemented in one single byte. The RX-events, are implemented in the event's second most significant bit. Concerning the TX-events, there are more aspects to take into consideration. The bits assigned to the TX-events are used not only to indicate that an application changed a property's value but also to give information regarding the number of retransmissions of the message to send as a consequence of this modification. The event's most significant bit is a control flag called SEND_NOW, activated as soon as the modification in the property value occurs, that remains active until the message gets transmitted. The least significant bits are used to count the message retransmissions, that must occur if the CM doesn't receive an Answer/ACK to a previously sent command in a given time period. This time interval is set in an auxiliary array, in which the retransmission timeouts are managed.

In order to store and arrange the indexes of the activated events, some auxiliary lists, called Lists of Ativated Events (LAE), were implemented in a single byte array. These lists are used in the management of all the events, regardless the property type. This approach was chosen in order to avoid the performance penalty of a full scan of the events' arrays every time one wants to check for activated events and to guarantee that these are treated according to the order by which they were activated. There are three types of LAE to take into consideration: Send_Now (that includes all the TX-events with an activated SEND_NOW flag and also the ones associated to messages to retransmit), Retransmission (contains the remaining TX-events, associated with messages that are awaiting to be retransmitted) and RX (includes all the activated RX-events). Each element of the list contains the index of another element, which is correspondent to the event that was activated immediately after. As an example, if cell number 2's value is equal to "1", this means that event number 1 was triggered immediately after the activation of event number 2. The elements considered to be the first (Head) and the last (Tail) of the list are stored in a pair of auxiliary bytes. The hexadecimal value 0xFF is assigned to any element that isn't in use or to the elements considered to be the last

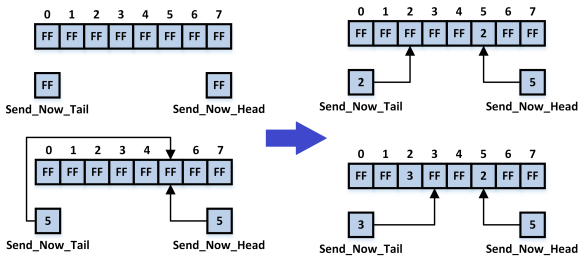

Fig. 5. List of Activated Events - Functioning Example

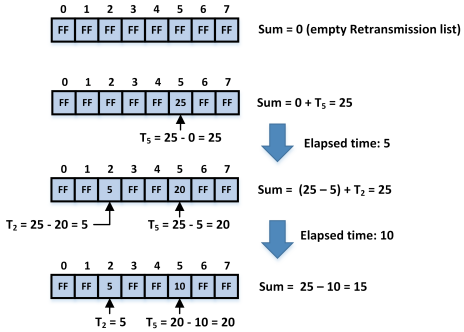

Fig. 6. Retransmission timeouts array - functioning example

entries of the respective lists.

To understand this reasoning, let's consider, as an example, that a certain CM controls a total of 8 properties, regardless the property type. In addition let's assume that, initially, all the events are disabled but, then, the TX-events 5, 2 and 3 are activated sequentially and placed in the Send_Now list. In Figure 5 is represented what occurs in the array as a consequence of these activations.

The retransmission timeouts, regardless the property type, are defined upon the activation of the event in a special array allocated for the effect. The contents of this array are managed by the MSG task in conjunction with the TIME task. In this array, time is measured in a relative manner. Each cell contains the number of time intervals that occur after the timeout associated with the event that was activated immediately before reaches zero. An auxiliary variable contains the sum of all the timeouts associated with activated events. If one wishes to set a retransmission timeout of 25 time units, and if the sum variable is equal to 20, the value to insert in the timeouts array is 5.

To get a better understanding of the principles applied, let's consider the example depicted in Figure 6. In this example, it was assumed that a retransmission timeout of 25 units of time is assigned to every TX-event upon entering the Retransmission list. Figure 6 represents what occurs in the time array when the TX-events 5 and 2 placed, sequentially, in the Retransmission list, assuming some time has elapsed between the events' activation.

The supervision of the retransmission timeouts occurs every time the MSG task is executed. In this process, as illustrated in Figure 6, the time elapsed between two consecutive executions of the MSG task is subtracted from the timeout value asso-

ciated with the head of the Retransmission list and from the "sum" variable. If the value to subtract is greater or equal to the head's timeout value, the respective event is removed from the Retransmission list and placed again in the Send_Now list.

F. Control Applications

The various sensors and actuators are directly managed by control applications. Given the DomoBus system context, this control is based on the devices' properties, that must be defined by the user according to the devices' characteristics.

An API was developed to allow a simple and intuitive interaction between the applications and the property values. This way, application programmers don't have to be concerned about the implementation details of the CM's software, along with the communication specifications, only having to understand the basic operating principles of the DomoBus system.

All the data structures, pointers and constants related to the properties and the events, along with some other configuration elements, are declared in a separate header file. The contents of this file are dependent on the control applications one wishes to implement in the CM and the number of devices and properties controlled by each one of them. The information needed for the generation of this header file is provided by the user/programmer of the Control Module, through a configuration text file. In this file, the user writes in each line all the information related to each control application.

A special program was developed to generate the header file with the CM configurations from the input text file. In addition, this program also generates in a separate folder the code and header files with a skeleton of the control applications. The created code files include some useful information, such as some constants, function macros, variable masks, and also basic versions of the init() and task() functions, to ease the application development process. The software's main file, with the cooperative multitask scheduler and a compilation batch file, that includes all the commands needed for the compilation and loading of the software to the board, are also generated automatically.

The help programmers in the development of applocations, is was developed an API which consists of a set of macros that allow the interaction with the property values using the names assigned to the applications and properties by the user in the configuration input file. There are four function types to take into consideration: **set_and_send(), send() get_new()** and **process()**.

The set_and_send() function is used to perform modifications in the property values. If this value is different than the one contained in the data structures, the CM will also send a message that indicates the occurrence of this modification. In a general way, this function is used in sensor applications.

The get_new() function, in turn, is used to get the details (type, ID and device number) of the actuators' properties that got their value changed by SET messages. With the obtained information, a programmer may access the property arrays in order to retrieve the desired value. This function is only used in actuator applications and is implemented in a separate file for convenience, as programmers should not perform modifications in this function. Calling this function is the first operation performed by the task() function of the respective application.

The process() function is used to treat a property value according to the data obtained obtained with the get_new() function. The process() function is essential for actuator applications to process property values modified by SET messages. This function may be used in sensor applications, albeit they aren't needed in this context, and therefore is treated as a normal function.

Finally, the send() function is used to activate a TX-event correspondent to a property value changed by SET messages after their treatment by the application that controls the respective actuators.

G. Gateway Implementation

Regarding the Supervision Level of the DomoBus system, it was developed a prototype of a Gateway application, implemented in a Windows PC, through which it's possible to test the software implemented in the Gateway Modules and Control Modules of a DomoBus Control Network. This application handles all the aspects related to the communication between the computer and the GM and also provides a simple user interface (UI) that allows the interaction with the devices controlled by the Control Modules through the computer's command prompt. With this UI, a user may easily interact with the devices controlled by the Control Modules of a DCN without having to know the implementation details of the DomoBus system, such as the structure of the Control Level data frames.

The Control-Level Gateway Module's software is composed by two layers, NET and SM_COMM, that are executed sequentially in a non-preemptive multitasking without context switches, the same way it happens in the Control Modules. The communication operations that occur in each one of these tasks are supported by two communication buffers, one for the storage of the messages received from the Supervision Module (SL-buffer) and one to store the messages received from the DCN (CL-buffer).

The control of the two communication buffers, performed by both software tasks, is based on a set of control flags implemented in two separate bytes, one for each buffer. Thee states are admitted for each one of these buffers: IDLE (is empty or contains no relevant data), BUSY (buffer is being loaded with data), and FULL (buffer contains a message that has to be transmitted).

Regarding the NET task, this is where occurs the exchange of data between the GM and the CMs of the DCN, through the nRF24L01+ radio modules. This task's operating principles are practically the same as the NET task of the Control Modules' software. The main difference relies on the fact that upon execution two operations are performed sequentially, namely the transmission of a message contained in the SL-buffer and the storage of a message received by the radio modules in the CL-buffer.

In the SM_COMM task occur all the data exchange operations between the GM and the SM to whom it's connected. In

this task, two operations are performed sequentially, namely the storage of a message received from the SM in the SL-buffer and the transmission of a message contained in the CL-buffer to the SM.

As aforementioned, the GM is connected to the PC used as a Supervision Module via USB. In the Arduino UNO board used as a GM, this connection is supported by an USB-to-Serial converter implemented in an ATmega16U2 microprocessor, connected to the ATmega328P via USART. The SM_COMM task handles the initialization and configuration of the ATmega328P USART, which is used to communicate with the SM. In this process, in which the USART is set to operate in asynchronous mode (UART), the most important factor to define is the baud rate, which is the rate at which the data is transmitted/received. In both the microprocessor and the USB-to-Serial converter, the available baud rate values depend on the frequency of the system clock. In this particular case, this signal is generated by a 16 MHz crystal oscillator that is also incorporated in the Arduino UNO board. At this frequency, one may apply a baud rate value from 2400 bps to 1 Mbps in both of them [7] [9]. Given these values, one must take into account the fact that the baud rate generators can't always perform an exact division of the oscillator frequency in order to obtain the desired baud rate. By analysing the microprocessors' datasheets, it's possible to verify that this situation won't occur if one sets a baud rate of 250 kbps, 500 kbps and 1 Mbps. It is desired that the transmission and reception of data occur as fast as possible. Therefore, 1 Mbps was the value chosen for the baud rate. At this baud rate, the transmission of a 32-byte message, given that each byte is composed of 8 bits, will last 256 μs , for example. This means that one can exchange, approximately, a total of 3900 messages per second, which is an appropriate value.

V. Results

A. Description of the Prototype

The developed prototype includes a PC as a Supervision Module, which runs the developed Gateway application, and 3 Arduino UNO boards. One of the Arduino boards was used as a Gateway Module and the two remaining boards were used as Control Modules. Each one of the Arduino boards is connected to a nRF24L01+ radio module.

Both CMs execute the same SYS application that controls a blinking LED based on the value of a property called "pulses". The number of blinks of the LED is equal to the value of the mentioned property. There was no need to implement more functionalities in the SYS application as the ones that were implemented already illustrate its operation principles.

The first CM, designated "CM 1", controls a green LED (LED_0) and a red LED (LED_1). Both LEDs are characterized by a single property designated "state" that controls the state of the LED. When one assigns a value greater or equal to 1 to this property, the respective LED will be switched on. Otherwise, when the property value is equal to zero, the respective LED will be turned off.

Regarding the other CM, designated "CM 2", this module controls a single LDR connected to the module's analog I/O port. This device is characterized by two properties. The first one, called "lux", corresponds directly to the value of light intensity measured by the LDR. The other property, called "darkness", is used to characterize the measured luminosity. The value of this property is equal to 1 in a dark environment and equal to zero on an illuminated environment. Naturally, the value of the "darkness" property is directly dependent on the value of the "lux" property and of what the programmer defines to be an illuminated or a dark environment.

B. Configuration Process

Taking into account the devices and properties to be controlled by each one of the Control Modules, two configuration files were generated. In both SYS applications, the "pulses" property is configured as an 8_BIT property.

Regarding CM 1, LED_0 and LED_1 are controlled an application called "LAMP". The "state" property that characterizes both LEDs was configured as an 8_BIT property. Concerning CM 2, the LDR is controlled by an application designated "LIGHT". The "darkness" property was configured as an 8_BIT property while the "lux" property was configured as an 16_BIT property, given that the Analog-to-Digital Converter converts the read light intensity into a 10-bit number.

C. Tests

Various tests were performed to validate the developed software, it's features and it's implementation.

In the first test a set of GET and SET messages were sent to the LEDs controlled in CM 1. The node sent back the expected Answer and NOTIFY messages. This way, it was possible to verify that one may interact successfully with values of properties controlled by the CM using GET, SET and NOTIFY messages, and also that one may control multiple devices with a single CM.

In the second test, the PC Gateway application was configured to stop sending the Answer/ACK messages to the received messages. This way, the Commands sent by the Control Modules would be retransmitted. For test purposes, it was defined that each retransmission would occur 1,25 seconds after the last one, and that a total of 7 retransmissions would occur. The results obtained confirmed the well-functioning of the message retransmission functionalities, such as the TX-events and their respective lists, along with the retransmission timeouts.

The third test was performed to verify whether the CMs were able to detect the errors in the received messages, such as invalid application numbers or property indexes, for example. In this test, a set of messages (one per each one of the 10 possible types of error) were sent to the Control Modules. The Answer messages obtained demonstrated that the Control Modules can detect errors in the received messages, sending back a reply with the correct error code, ensuring a robust interaction.

Next, it was performed a fourth test to demonstrate that the Control Modules were also able to manipulate D_ARRAY properties. In order to create conditions for the performance of this test, CM 2 was reconfigured with a special application that

changed the value of the ”pulses” property controlled by the SYS application according to the dimension of the received D_ARRAY properties. This way, the number of blinks of the LED controlled by the SYS application would change to match these values, as verified.

Finally, it was performed a final test to demonstrate that the properties of the various devices controlled by Control Modules may be used in the development of automatisms in the Supervision Level. In the PC Gateway application was implemented a simple automatism, in which SET messages were sent to the ”state” property of LED_0 according to the value of the ”darkness” property of the LDR controlled in CM 2, received through NOTIFY messages. The results obtained were the expected.

VI. Conclusion

This article presented the work performed regarding the development of Control Modules (CM) and Gateway Modules (GM) for the DomoBus home automation system.

In the DomoBus approach, domotic devices (sensors and actuators) are treated as generic entities characterized by a set of properties. One may interact with the property values through a set of control messages: GET (obtain property value), SET (modify property value) and NOTIFY (report a change in a property value). With this simple model, it is possible to treat home devices with the most diverse characteristics according to the same principles.

The main objective of this work was the development of CMs. These components are directly responsible for the control of home devices according to the values of their properties. The CMs are able to control multiple devices of different types simultaneously, which is an especially useful feature for houses with a high number of devices to control. These modules are based on low-cost generic boards, similar to Arduino boards. To keep the price as low as possible it was decided to use boards based on 8-bit microprocessors, which have some hardware limitations. To facilitate the implementation of a prototype, Arduino UNO boards, the simplest of the Arduino product line, were chosen. The software developed for a CM is based on a simple and modular software architecture, composed of three main components: NET, MSG (MESSAGE) and PROP (PROPERTIES). These components, along with the applications responsible for the control of domotic devices, are executed sequentially in a non-preemptive operative nucleus in a ”round-robin” fashion. This simple solution was chosen for being well-supported by the used hardware, for allowing the development of the software modules independently and for facilitating the conjunction of the different software modules.

One of the main objectives of this work was to facilitate the development of applications for the Control Modules. As such, it was created an auxiliary program for the configuration of the CMs, based on a set input data inserted by the user, such as the application names, number of devices, number of properties of each type (8_BIT, 16_BIT and D_ARRAY), and the properties' names. This program generates a header file where the PROP component's data structures are declared. In addition, this configuration program also creates a set of code files which include the basic skeleton of the applications to develop. It is also offered an API through which programmers may access and modify property values using a set of macros that use the application and property names inserted in the configuration process. With the defined architecture and the developed software, programmers don't have to be concerned about the module's communication details. They just have to deal with the control of the home devices based on their properties' values.

It also was developed the software of the Gateway Modules of a DCN, that interface the Supervision Modules (SM) of the Supervision Level, the Control Modules of a DCN. The developed software is based on a modular software architecture composed of two components: NET, where occurs the exchange of packets with other nodes of a DCN, and SM_COMM (Supervision Modules Communication), responsible for the communication between the GM and the Supervision Modules.

Finally, it was also developed a prototype to test the functionality of the developed modules. This prototype included a PC as a Supervision Module, a GM and two CMs, that controlled their own set of devices. The results obtained of the various tests validated the proper functioning of the implemented functionalities and the software architecture.

In the future, it should be developed a process for the insertion of the skeletons of the Commands sent by the Control Modules data into the EEPROM. Additionally, it would be rewarding to use other devices as communication modules, instead of using only nRF24L01+ radio modules, to demonstrate the modularity of the developed software architecture and, in particular, of the NET module. Concerning the CM's configuration software, some improvements should be performed to prevent some properties from being controlled by SET messages, especially in the sensors domain.

References

[1] G. Almeida, “Domótica - Tecnologia do Futuro ou Tecnologia para o Futuro?,” *A Domótica e a Casa do Futuro, AveiroDomus - Workshop Domótica*, Aveiro, 2006.

[2] R. Nunes, “DomoBus - The User In Control,” *A Domótica e a Casa do Futuro, AveiroDomus - Workshop Domótica*, Aveiro, 2006.

[3] KNX Association, “KNX Basics.” https://www.knx.org/media/docs/downloads/Marketing/Flyers/KNX-Basics/KNX-Basics_en.pdf, (last access: January 2017).

[4] Echelon Corporation, “Introduction to the LonWorks Platform - Revision 2.” http://www.echelon.com/assets/blt893a8b319e8ec8c7/078-0183-01B_Intro_to_LonWorks_Rev_2.pdf, (last access: January 2017).

[5] F. X. Jeuland, *La maison communicante - Réussir son installation domotique et multimédia*. Éditions Eyrolles, 2012.

[6] Echelon Corporation, “Neuron C Programmer's Guide.” http://www.echelon.com/assets/blt63fb8a189770fe78/078-0002-02H_Neuron_C_Programmers_Guide.pdf, (last access: January 2017).

[7] Atmel Corporation, “ATmega328P Datasheet Complete.” http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, November 2016.

[8] Nordic Semiconductor, “nRF24L01+ Single Chip 2.4GHz Transceiver Product Specification v1.0.” https://www.nordicsemi.com/eng/content/download/2726/34069/file/nRF24L01P_Product_Specification_1_0.pdf, September 2008.

[9] Atmel Corporation, “8-bit AVR Microcontroller with 8/16/32K Bytes of ISP Flash and USB Controller.” http://ww1.microchip.com/downloads/en/DeviceDoc/doc7799.pdf, September 2012.