Smart Behaviours for Smart Homes

José Augusto de Góis Rodrigues de Sá

October 2019

Abstract

Nowadays home automation systems represent an important market. Everyone wants to have the power to control their home since these systems offer comfort, security and good support for energy management. There are a variety of home automation systems available in the market and, although there has been evolution related to the interaction between the user and the system, most of them do not offer good solutions regarding flexible, generic, and powerful automation mechanisms that can be applied in a simple way.

In this work a solution will be presented, in the context of the DomoBus system, which uses the concept of "Automation Block". Automation Blocks are software entities that have various inputs and outputs and perform some predefined function. Automation Blocks can be very powerful and, because of that, quite complex. So, a web application was developed that fully supports the development of Automation Blocks, which is expected to be done by knowledge users. Those Automation Blocks can then be shared and applied to real systems. This last task can be performed by common users, which just have to select which Automation Blocks to apply and associate real devices to their inputs and outputs. This is also supported by the developed application, allowing users to easily customize the behavior of their homes and benefit the most from their home automation systems.

Keywords: DomoBus, DomoBus Automation Block, Home Automation system, Home automation devices, Automation mechanisms, Ruby On Rails, Vue.js

1. Introduction

The desire to control devices in our home or even remotely program them to execute a specific task at a given time is a growing interest nowadays. People are constantly searching for ways to ease everyday tasks and the possibility of control and manage resources.

Home automation systems allow users to take control of their homes and perform several tasks. With this type of system, the user can easily control his house from, for example, a smartphone.

2. Problem

There is a big variety of solutions of home automation systems like Insteon, Amazon Echo, IFTTT, but very few, or none, allow the user to have a fully automated home. When we look to what the market has to offer, most systems allow the user to create scenarios and execute commands via smartphone, voice, among others. Some systems offer fully automated tasks but only when integrated between themselves.

3. DomoBus

DomoBus is a home automation system created by Professor Renato Jorge Caleira Nunes. It was developed as an academic project with the goal of being a simple but extremely powerful an expandable system [10].

DomoBus consists essentially of Console Modules and Supervision Modules that communicate with each other over a network. A console module is a simple processor that can connect to power lights, movement sensors, thermostats, among others, and communicates directly to only one supervision module. A supervision module controls and, like the name says, supervisions the system. It receives rules, defined by the user, and use them to process requests received by the console modules and send commands as response. The proposed solution will be done on the supervision module.

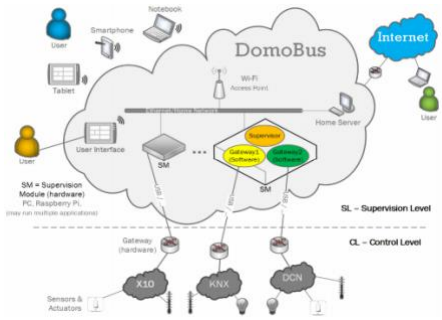

Fig. 1: DomoBus Supervision Level [13]

There are 3 type of messages that write (SET), read (GET) or inform of a new value (NOTIFY) of a device that is included in the DomoBus Protocol. Every DomoBus system is specified by an XML-based document divided in two parts [11]. One that allows the specification of devices and their properties and other that specifies the house and all their components. XML allows the user to define generic applications that read it and fit the system. The hierarchy accepted by the XML is House > Floor > Division > Device. Every device is a generic entity that is characterized by properties depending on the device. For example, if we have a light, we can have two properties, one that turns the light ON and OFF and other property to control the brightness of the light. Each property has a value that can be one of three types. A **ScalarValueType** that it is an integer that can be represented by 8 bits (from 0 to 255) or 16 bits (from 0 to 65535), an **EnumValueType** that it is a pair of [name, value] and the value is represented by an 8-bit quantity or an **ArrayValueType** that, like the name says, it is an array of bytes.

In a low-level a device is identified by a number with 32 bits and the properties by a property descriptor with 8 bits which is divided in 3 sections: bits from 0 to 4 represent the property ID, bit 5 represents an invalid flag and bit 6 and 7 represent the property type. The property type can be: 8_BIT (b7=0, b6=0, value=0), 16_BIT (b7=0, b6=1, value=64), 32_BIT (b7=1, b6=0, value=128) and D_ARRAY (b7=1, b6=1, value=256). The D_ARRAY is a DomoBus Array that has an additional byte that indicates the size of the that is useful inside.

However, in a high level they are defined with XML following defined patterns.

4. Proposed Solution

DomoBus Automation Block (DAB) is an automation block, already documented by Professor Renato Jorge Caleira Nunes, that can be programmed to perform almost any task that a user desire. The block receives a series of arguments that are processed using state machines and execute a result. The arguments are defined as inputs and the result as outputs, allowing the definition of complex rules of automation [9].

Finally, a block has one or more states. A state is composed by one or more transition that its divided into two components: a condition and an action. One or more actions are triggered when a condition is met. When this happens, the transition is executed. The actions executed by the transition ban be of two types: local or normal. A local action is responsible for changing the current state and manage timers by starting or cancelling them. A normal action only deals with the output changes. Its responsible for changing the values of the outputs.

A block and all its components are defined in text file following some rules that will be explained next.

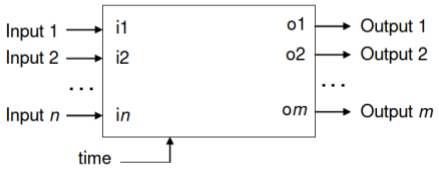

Fig. 2: Representation of a DomoBus Automation Block [9].

4.1. Input aggregation

An automation block can allocate several properties of one type by using combinatory logic with AND or OR operator. The AND operator means that an input is true only if all properties connected are activated, The OR property means that only one property activated is enough to activate the input.

4.2. Default inputs

Inputs are allowed to be left open and without any connection. The value of the default input should be 1 or 0 equivalent to true or false respectively.

4.3 Input type

Like mentioned above, each input of a block is connected to a property type. Those types are represented by the values 0, 64, 128 and a 192 (DomoBus Array). Some properties can work in a mode called “action”. When working in this mode, a property, when activated, triggers an action and then returns to is initial state. When a property is signalized as an action, then it should add 1 to its type so becomes 1, 65 or 129. A DomoBus array can’t work in action mode.

4.4. Input condition

An input can receive any value if it agrees with its type, but the internal logic of a block only operates with the Boolean values, true or false. Because of that,

an association is added to every input with the purpose of converting them to a Boolean value. The operators available to define those associations are EQ-equal, NE-not equal, LT-less than, LE-less or equal, GT-greater than, GE-greater or equal, DO-true for any no value and INV-true if the value is invalid.

4.5. Input definition

After explaining the main features of the inputs, now will be demonstrated an example of how define them.

i0 0 GT 50 OR 0 - The following example indicates an input that is identified by the number 0 and has type 0 (8 bits) which means it is an enum or a scalar. The properties will be tested to see if they are greater than 50 and return the respective Boolean value. If there are more than one property of the same type, those properties will be aggregated using the OR operator that will return the value of that input (true/false). If no property is defined, the default value will be 0 which means false.

4.6. Output definition

An output starts with letter 'o' receives and id and a property type that must also be 0, 64, 128 or 192. Example:

o0 0 – The following example indicates an output that is identified by the number 0 and has type 0 (8 bits) which means it is an enum or a scalar.

4.7. Timer Definition

A timer is defined with a letter 't', an id and an integer value that represents, in seconds, the time that the timer waits. Example:

t0 10 – The following example indicates a timer that is identified by the number 0 and the time value of 10 seconds.

4.8. State Components Definition

As mentioned earlier, a state consists in one or more transitions that are divided in conditions and actions. The definition of the state components is done hierarchically and will be explained next.

4.9. Condition definition

A condition is defined with the letter 'f', an id, an operator and one or two operators depending on the operator. The operator's EQ and NOT only accept one operand. The condition represents a logical expression and uses the Polish prefix annotation. In this annotation the operator precedes the operands. The operands of a condition can be any of the properties of the input as well as other previous conditions.

As mentioned previously, a group of conditions represents an expression. An expression is defined with a letter 'e', an id, and the conditions. Example:

e0 f1: OR *x y* f2: AND f1 *w* f3: NOT f2 – The following example indicates expression that is identified by the number zero and has 3 conditions: f1, f2 and f3. The first condition is identified by the number one and represents the Boolean value of the property *x* OR the property *y*. The second condition is identified by the number two and it is an example using a previous condition as an operand. Represents the Boolean value of the first condition AND the property *w*. Finally, the last condition, represented by the number 3 uses only one operand because uses the NOT operator. It represents the negations of the Boolean value of the second condition.

4.10. Action Definition

An action is not defined by any letter or id and as mentioned previously, it can be local or normal. An action represents the definition of locals and normal actions, each one, ending with a single '.'. A change of state and starting or cancelling a timer are local actions. The change of state is represented by the letter 's' and the id of the state it will be changed. Starting a timer is represented by the letter 't' and the id of the timer. Cancelling a timer is represent but the letter 'c' and the id of the timer. Changing output values is a normal action and its represented by a letter 'o', the id of the output, the equal symbol '=' and the value that the output will be changed to, 0 or 1. Examples:

s3 c2 . o1=0 . – The following example represents two local actions and one normal action. The first local action indicates that it will change to the state with the id 3 and the second indicates that the timer with the id 2 will be cancelled. The normal action indicates that the output with id one will turn false.

4.11. State Definition

A state is defined by the letter 's', an id and all the components that will define it in a hierarchically form. Before the definition of the states, a line is reserved to list the ids of every states. The line starts with capital letter 'S' following the two dots symbol ':' and the ids of the states. Example:

```
S: 1 2
s1
    e0 f1: OR x y f2: AND f1 w
        s3 c2 . o1=0 .
```

s2

e0 f3: NOT f2

. o1=1 .

The following example represents two states. The first line declares the states that will be defined next. The next lines are just the combination of the definitions previously explained with the exception of the line that indicates the start of each state. In this example we define two states with the ids 1 and 2.

4.12. Instantiation Process Definition

The instantiation process is what generates the final file that has the connections between the block and devices. A text file is generated with several commands when those connections are made. The file is called a template and will be explained next.

The file starts by identifying the block by using the letter 'u' followed by its id. Next the inputs of the block are associated with the device's properties chosen. Each line defined one input and starts with a letter 'i' followed by its id. Next the properties that are linked to the input are each one defined in 3 numbers separated with commas. The first number is the id that identifies the device which the property belongs to. The second number identifies the property descriptor that's was explained in section 3. The last number identifies the property configuration. This property is different between properties types. For 8_BIT, 16_BIT and 32_BIT it represents if the property acts in normal (0) or action mode (1). In case of a DOMOBUS_ARRAY property, the last field represents the size of useful information present in the array. The output definition follows the same analogy that the inputs. The file must always end with the letter 'e'. Example:

u5
i0 10,0 1 15,1 0
o2 12,0 0
e

In this example we have a template for a DomoBus Automation Block with id 5 that has one input with two properties associated and one output with one property connected. The first property linked to the input is from the device identified by the id 10, has a property descriptor of zero which means that has an 8_BIT property type and has 1 as property configuration which means the property works in action mode. The other input and the output follow the same analogy. Finally, like mentioned before, the file ends with the letter e.

4.14. Solution

All the points previously explained are powerful and allows the user to have a very good control of a system. The downside is that, the way that all the system is defined right now is not usable at all for a normal user and very difficult even for users comfortable with programming and with the system.

The solution consists in an interface that would allow users to create and manage different types of blocks in their homes in a very usable way. A simple and beautiful interface with simple steps that encapsulates all the complexity of creating and managing a DomoBus Automation Block to allow any user to enjoy the system.

This solution will be divided in 3 parts: a server, a database and an interface. Defining and creating DomoBus Automation Blocks is not easy and because of that, the interface should guide the user thought the full process, avoiding errors and performing validations.

The technologies to be used are Ruby on Rails for the server, Vue.Js for the interface and PostgreSQL for the database.

5. Implementation

In this chapter it will be described all the parts of the application that was implemented to allow users to create Domobus in a very simple way and to use them to connect to different devices present in their homes.

5.1. Architecture

The Domobus system is composed by two components. One web application made with Vue.Js and an API made with Ruby On Rails. The application sends requests to the API that, after certain validations that will explained later, returns a response to the web.

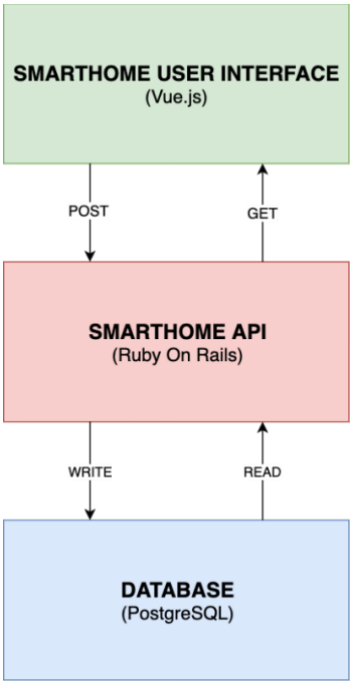

Fig. 3: System Architecture.

5.2. Users

The system has three type of users: User A, User B and Admin. User A is the "normal user" that doesn't have knowledge of how the system works and that is why is important that the user interface and experience must be as good as possible so the user can add a house to the system and use DABs present in the server to create their automated actions without much difficulty.

User B is the type of user that has knowledge of the system and how the DomoBus Automations Blocks work and how to program and create them. He can also add houses but besides using DABs in the server, the User B can create and add DABs to the server himself.

The Admin is the superuser of the system and is the user that decides if a user is the type A or B. He can give and revoke permissions for any user as well revoke any DAB that, in his opinion, doesn't seems correct or useful.

5.3. Authentication

To manage authentication, Json Web Token (JWT) is used. When logging in, a unique token is generated using a secret and assigned to the user with a 24-hour validity. The validity of the token can be changed for the desired value. Then, the token is sent to the browser where it is stored and added to the authorization header every time a request is sent to the API. Before every action, the API verifies the validity of the token and retrieves the user information from it. If the user information is correct and correspondents to the current user, then the request goes through. If not, the API returns a 401 error to the browser and the user is logged out.

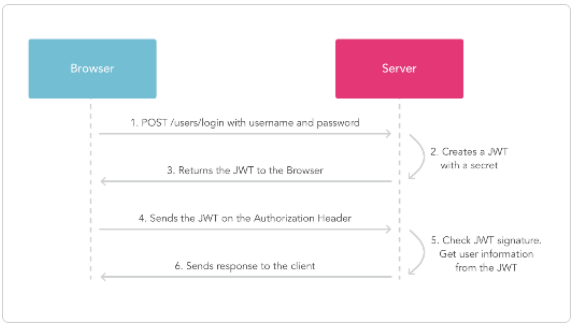

Fig. 4: JWT diagram [14]

This error indicates that the action executed was unauthorized. This method protects the system against any attacks with the objective of forging the token.

5.4. XML File

A user can add is house using an XML file that must contains all the information required by the API to perform the action. If the information provided in the XML is not valid, a toast error is presented to the user in the interface. XML stands for E**X**tensible **M**arkup **L**anguage and it uses tags to define objects. Those objects can have child objects and so on [14]. Below it is presented an example of an XML file.

```
<House ID="1" Name="Jose's Home" Address="R. Latino Coelho 24, 1050-132 Lisboa" Phone="219321243">
  <FloorList>
    <Floor ID="1" Name="Ground floor" HeightOrder="0" />
  </FloorList>
  <DivisionList>
    <Division ID="1" Name="Kitchen" RefFloor="1" />
    <Division ID="2" Name="Living room" RefFloor="1" />
  </DivisionList>
</House>
```
Fig. 5: XML example file

5.5 Adding a house

If a user desires to add a house, a file must be chosen from the submit file area that is present in the Home view of the system. As mentioned before, the file must have the XML format and because of that, the user isn't allowed to submit a different type of file because a validation is made directly on the file input. This prevents the user to mistakenly select another type of file.

After a file is chosen, the user must submit it. This will trigger a method that will send to the API a json object with a *formData* that contains the file. In the API side, the XML file is processed [16]. The file is iterated and when an XML object is found, its type and tags are detected, and the object is saved in the database accordingly to what was found before proceeds to check its children.

5.6. Creating a Domobus Automation Block

Like mentioned before, users of type B can create DAB's. The creation is made from the Domobus

creation workflow that has 5 steps, DomoBus Automation Block, Inputs, Outputs, Timers and States, that will be explained below. It will be used an example of automated light using a movement sensor. In the Fig. 6 is possible to see the state machine diagram of the example.

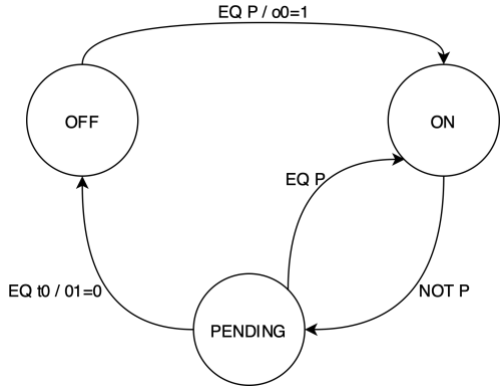

Fig. 6: State machine diagram.

The DomoBus Automation Block step is where the user chooses a name and a small description to the block. It must be self-explanatory and straight forward so that the users that decide to use the DAB don't have any doubt of what it can and should do.

In the Inputs step the user adds all the inputs that the DAB accepts. The inputs will be connected with the properties of any device (sensors for example) to trigger an action. For each input, the user chooses a name, a property type, a mode, the operator, the value, the condition and default value in case the block is used without this input.

It is possible to add and delete more inputs. This shows that system is flexible and scalable. The design implemented was also chosen to allow the adding of new objects to be user friendly.

In the next step it will be defined the outputs by choosing a name and a property type to each one. The outputs will connect to some kind of external source like for example a lamp, an AC, etc. The actions triggered in the inputs may or may not affect the outputs. Just like the inputs, it is possible to add multiple outputs using the button "New Output".

Setting the timers is the next step. This is the easiest step because the user only needs to add a timer and define its time in seconds. The user can add as many timers as he wants.

The final step is the hardest one and it is the one where the user must use his programming skills and his knowledge of the system. In this step is where the user creates the states, define the transition(s) required to change from a state to another and the action(s) triggered when changing state. A state must have a name and one or more transitions that are divided in conditions and actions. The conditions together make an expression. When a condition is met, the respective actions are triggered. A state represents all the ways an output can be represented. In Fig. 7 it is possible to see the first state of a very simple automated light block.

Fig. 7: First State - "Off"

In the first state, there is one transition with one condition and two actions. The name of the state is "Off", and it is the first step. When the sensor, that is represented as i0 (abbreviation for input 0), is equal which means that it's true, the two actions are triggered. The first action is local and changes from the state 0 "Off" to state 1 "On" and the normal action sets the output to one which means the light will be turn on and stay that way.

In the second state, the light is on so that's the name it has. In this state we have again one transition with one condition and two actions. When the input 0, the sensor, does not detect any movement the timer will start and the state will change from "On" to "Waiting", the third state.

Fig. 8: Third state "Waiting"

In the "Waiting" state is when the light is still on, but the timer is running because the sensor didn't detect any movement. At this point there can be two outcomes and that is why this state has 2 transitions. In first transition if the timer equals, which means it is

true and it has expired, then the timer will be cancelled, the state will be changed from “Waiting” to “Off” and the light will be turn off by changing the output value to 0. In the second transition however, if while the timer is still running the sensor detects any movement, then the timer will be cancelled, and the state will change again from “Waiting” to “On”.

5.6. Write the block to a file

After the user finishes defining all states a request is sent to the API. The request contains a JSON object with all the data created during the DomoBus Automation Block creation workflow and it is organized hierarchically. This means that when defining a state object, a transition object will be defined inside it, and the actions and condition will be defined inside the transition object and so on.

The object is parsed, and all the valuable information is used to create all the components that together creates a DAB. That information is saved in a database in form of an Input, Output, Timer, Transition, Condition, Action and State (Fig. 10). Then, the DAB is written to a file following the rules defined by Professor Renato Nunes in the Domobus Automation Block documentation [9].

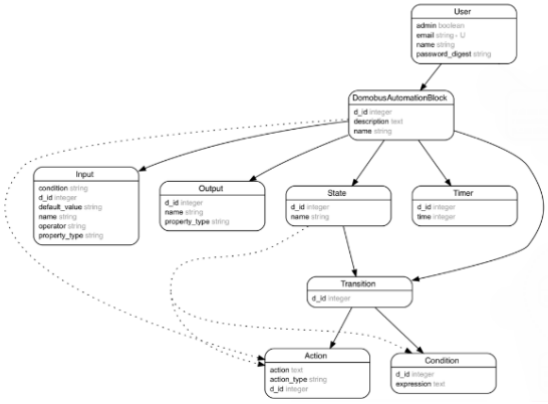

Fig. 9: DomoBus Automation Block Entity-Relationship Diagram.

5.7. Editing a Domobus Automation Block

The ability of editing a DAB is actually one of the most important features of the system and for several reasons. It allows the possibility of correcting any mistakes that its creator committed without having to create a new one from scratch and to think again on all the complexity required to create it. It also allows the possibility of extending it and improve it at any given time so the DAB can stay updated with the requests desired by the users.

When designing the process of editing a DAB there were some things that were important to reflect. The usability of the method was important so that a user could edit the block easily and without to many constraints. It was important that the information of the block was shown so that the user could see the current state of it before making the desired changes. For all this reasons, and because the create method is very clear and simple to use, it was decided that make the user be able to edit a DAB by reusing the steps of creation was the best option because not only it has a good usability but also because it avoids the necessity of learning new workflows for very similar methods.

To edit a DAB, it was added a pencil icon to every block on the list. This icon is common symbol to edition of a certain object.

There were some things that needed to be changed on the create algorithm so that the integration of edit functionality could be done successfully. Because the create and edit respective buttons will send the user to the same view, it was necessary to give the system some kind of information if the user wanted to create or edit a block. When the user starts the update process, the id of the block he wants to edit is saved in local storage as well in the vuex. The local storage is a web storage present in the browser that give websites the possibility of save information without expiration date using JavaScript. Vuex is a library of the Vue.Js framework and works as a centralized storage where it is possible to data as state between different views [23].

By saving the id of the DAB, it is possible to have access to it in the new view and to perform the API call to get the information about the DAB. If the id, for some reason is not recognize and the server does not find any block with that id, an error message is provided to the user. When the result arrives, the message is processed to fill the view and display the current status of the DAB. When the rendering process is finished the user can start modifying the block by deleting, changing or adding information to it.

In Rails, when a model object is updated, in this case the DomoBusAutomationBlock, the server knows what fields are different and automatically updates those field when the update method is called upon themselves. But the problem with this update is that, besides updating the information of the block, it can also be updated the relationship models that are connected to it (Input, Output, State, etc.). In order to the API to know what models where modified in front-end, some changes needed to be made on the *create/edit* view.

The first problem with this solution is that there is no information that can make the API identify a specific input. This approach faces no challenges when we are talking about the create method because an

object model has an identity not when it is instantiated but only after its creation. When a block is created from scratch, all the objects models are being created so there is no need of identity information but when it is updated it is a different story because the server needs to know what inputs were modified. To solve this problem, it was added an id attribute to each input. Besides identifying the input, this attribute also fixes the problem of distinguish the news inputs with the existed ones. On the server side, when iterating over the inputs array, a verification is made with the id of each object. If the id attribute is present and it is not empty, then the serve knows that it is an input that already existed, so it checks for updates and execute them. If not, it creates a new input with the corresponding information.

There was one last challenge regarding the update: how does the server know when an input is removed? This was solved by creating a method that sync the data present in the database and the data received. The method iterates over the array of data and checks if every id in the database is in the array. If one of them is not in the array, then the servers knows that it was deleted on the front-end and deletes the record from the database. The objects without id are ignored.

Although it was used the Input as example, the algorithm is valid to every model.

Some problems were faced during the implementation, most of them because the lack of organization of the edit algorithm. The "create" algorithm was already made so it was hard to find a way of implementing everything together without breaking everything.

5.8. Connecting house´s devices to a Domobus Automation Block

Allowing the user to use a DAB made by other user and that is present in the database to connect to a device in house is one of the last steps of the workflow. It will be explained how the connections are made, the steps taken to prevent the user to mistakenly link properties that are not supposed to be and how the file with the connections is created and generated.

Designing the functionality of this feature was one of the most challenging. Linking objects in a user interface is not easy to execute and to make it easily usable. In a first approach, an algorithm of drag and drop was made. This method consisted in dividing the view between a device and a block. The block and device would have boxes that would represent the input and output for the block and the properties for the device. The user would drag the properties from the device to the respective box on the DAB. Later on, this solution was cut off because of complications related to scalability and flexibility of the method. Other reason was the usability of the system. Because the system allows the block to be connected with properties from different devices from different divisions, the amount of information needed to provide to the user when constructing the connection between several devices and a DAB, it was almost impossible to have a clean and usable algorithm of drag and drop without giving up on some part of the crucial information. The solution chosen and present in the system is one based on html <*select*> element. This solution faces all the previous problems in a very clean way and will be explained below.

The information needed to be shown to the user related to the device is its name, the division where it is, its properties and the type of the properties. To connect the block the user needs access to a list of public DABs and, when one is selected, the information about it, its inputs and outputs and the properties of the same. As mentioned earlier, there is a lot of important information that needs to be presented to the user in the most usable way. Next it will be explained how that was done and the connections made.

The houses added by a user are present in the house list on the Home view. When the user clicks on the name of one of them, his redirected a view with the information of the selected house. This is the first step of the workflow that is being explained and the first piece of information needed to present to the user. It is possible to see that every division is shown on the respective floor. The user can enter a division by clicking in one of them.

Like mentioned previously, when making the connection between a DAB and a house, is important that the system has the flexibility that allows the user to switch between divisions without losing the properties already assigned to the block. In order to do this, it was needed, inside the division's view, a way of changing between divisions without going back to the house view. This was accomplished by joining all the divisions and respective functionalities in one large object and set up an arrow in the top corner of the properties container to move between them. The division view is divided in a container on the left side that encapsulates the divisions with their devices and respective properties and another container on the right side that contains the information about the DomoBus Automation Blocks.

It is possible to choose between all the blocks that exist in the database and that were made by other users.

When choosing a block, a container and the save button appears. From this container is where the user can actually make the connection between the properties that are inside each device present in a division and the inputs and outputs of a block. The user is presented with two different tabs, one aimed for inputs and the other for the outputs. When selecting one of the tabs, the inputs/outputs are listed in the form of a block. Inside the block is listed every property that exists in every device present in every division.

As previously mentioned, an input can only be linked to properties with the same type and to prevent users from mistakenly choose one that is not, the input only lists the properties that are allowed to connect to them which means they have the same type. For a better usability and to give user some feedback about which property is selected, when passing the mouse over the property, the container of the correspondent property on the left side turns green. This allows the user to have a better perception of which device the selected property belongs to.

Other functionality that the algorithm must support is to make an input accept several properties from the same or different devices. This is a key feature of the DomoBus system and is what makes it so powerful and flexible. This was accomplished by making the making the block accept several clicks and properties. The properties selected are then pushed to an array that is displayed in above the bellow the name of the input. If a user desires to disconnect a device's property from the input, all he needs to do is click on the arrow that appears on top-right corner of the properties that are displayed in the array. In the Fig 11 is possible to see all the functionalities, mention above, at work.

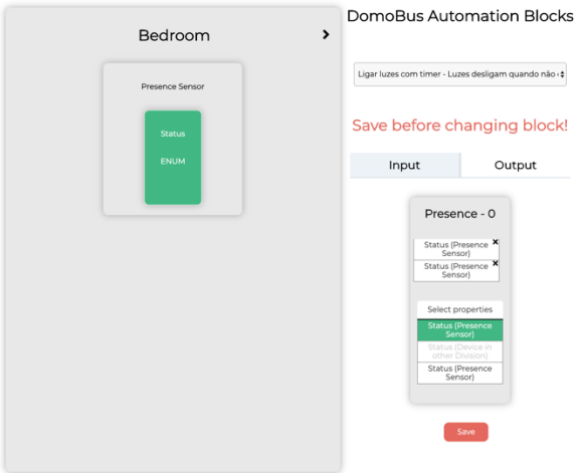

Fig. 10: Connection between a block and devices.

6. Evaluation

In order to test the interface and its usability, two different tests were conducted. In both tests there were steps to follow and a scale where the user should identify the difficulty when executing the step. The scale was from 1, meaning the task is not usable at all, to 5, the task has a very good usability. It was also registered the number of wrong clicks the user took before making a correct one. It will be detailed an example for each type of user.

In the test designed for the user A, there were thirteen questions. From all the thirteen, one question received a 3 grade, two received a 4 grade and eight received a 5 grade. The lower grade was awarded to task of adding a house with the justification that he did not had previous knowledge of XML and had a hard time understanding the concept of it. The user made one wrong click in the tasks 'Verify the Inputs and Outputs of the DAB', 'Connect a Device to an Input' and 'Connect a Device to an Output'. It was asked to the user to make a final comment about the system. The user gave a grade of 8. The user liked the design and general functionality of the system but also commented that the interface must have more labels that explain what each component does.

In the test designed for the user B, that has permissions to make blocks, there were seventeen questions. From all the seventeen, one received a 2 grade, four received a 3 grade, five received a 4 grade and seven a 5 grade. The lower grade was awarded to the step of adding an input to the DAB, the first step of creating a block. The reasons behind it was due to two factors: the first contact with the creation workflow and the poor label for properties type. The user felt an initial confusion when started the process of creating a block that was later dissipated when he got more used to the workflow. The other cause was the fact that the properties types are defined in 8_BIT, 16_BIT, 32_BIT and DOMOBUS_ARRAY which the user thought that, even for the users that have some knowledge of the state, are not user friendly at all. For the final comment the user gave a grade of 8. He said that the system is well design and very usable in general and that it facilitated a lot the work for DAB creators. He also said that some actions could have labels with small text explaining their purpose.

7. Conclusion

Home automation systems are growing in number and features.

The Smart Homes web application allows user to have control of their system by using DomoBus Automation Blocks. The system encapsulates most of the complexity behind the creation and management of

the blocks allowing almost every user to be able to use it in a simple way. The flexibility that the DAB presents offers the possibility of creating logic that can be reused for several devices as long they have all the same property type.

Even though the application allows the user to perform the core functions of the system, there are still several work that can be done in the future in order to optimize it. Allow the user to manage is houses by manipulating them directly in the application or even deleting them. This prevents the user from manipulating XML file and having to update it every time he wants a new division or device. Allow the user to check which block he is using and where he is using it. Create even better ways to manipulate the complexity of the DomoBus Automation Blocks by, for example, design a more user-friendly way of displaying the different properties type to the user.

DomoBus gives power to the user to create their own automated tasks using DomoBus Automation Blocks and they can do that using the Smart Home application.

Technology is becoming an integral part of our life and the next step is letting it enter our homes. The future holds many secrets, but one thing is for sure, Smart Homes is the smart way to live.

8. References

[1] Lumpkins, W.: Home Automation: Insteon (X10 Meets Powerline). IEEE Consumer Electronics Magazine, 140-144 (October 2015).

[2] What is Insteon?, http://www.rfwireless-world.com/Terminology/Insteon-basics.html, last accessed 2018/12/8.

[3] Carbonette, B.: Connecting Arduino to IFTTT. In: Hackster.io (April 2017).

[4] Insteon page, https://www.insteon.com/, last accessed 2019/1/3.

[5] Amazon Alexa page, https://developer.amazon.com/alexa/connected-devices, last accessed 2019/1/4.

[6] ZigBee explanation, https://internetofthingsagenda.techtarget.com/definition/ZigBee, last accessed 2018/12/27.

[7] Smart Home Skill API page, https://developer.amazon.com/docs/smarthome/understand-the-smart-home-skill-api.html, last accessed 2019/1/5.

[8] Amazon Echo auto page, https://www.amazon.com/Introducing-Echo-Auto-first-your/dp/B0753K4CWG, last accessed 2019/1/6.

[9] Nunes, R.: DomoBus Automation Blocks, Programming the DomoBus System, Technical Report, Instituto Superior Técnico, University of Lisbon, 2018/09/16

[10] Nunes, R.: The DomoBus System Specification Language, Programming the DomoBus System, Technical Report, Instituto Superior Técnico, University of Lisbon, 2018/09/16.

[11] Nunes, R.: Especificação XML de um Sistema DomoBus, Technical Report, Instituto Superior Técnico, University of Lisbon.

[12] Welcome Home, https://ifttt.com/applets/kCYK8bu3-welcome-home?term=welcome%20home, last accessed 2019/1/7.

[13] Nunes, R.: AI_03_Domotics_DomoBus, Instituto Superior Técnico, University of Lisbon.

[14] JWT Diagram, https://cdn.auth0.com/content/jwt/jwt-diagram.png, last accessed 2019/1/5.

[15] XML.: https://fileinfo.com/extension/xml, last accessed 2019/1/5.

[16] Evans, J.: REXML: library for Ruby On Rails, https://github.com/ruby/rexml, last accessed 2019/1/5.

[17] IFTTT.: API, https://platform.ifttt.com/docs/api_reference.

[18] SmartLabs Technology.: INSTEON_Developers_Guide, http://cache.insteon.com/pdf/INSTEON_Developers_Guide_20070816a.pdf (August 2007), last accessed 2019/29/9.

[19] Stringify is on IFTTT, https://ifttt.com/discover/stringify-is-on-ifttt, last accessed 2019/10/5.

[20] Build Skills with the Alexa Skills Kit, https://developer.amazon.com/en-US/alexa/alexa-skills-kit/resources/training-resources/alexa-skill-python-tutorial, last accessed 2019/10/6.

[21] Understand Custom Skills, https://developer.amazon.com/docs/ask-overviews/build-skills-with-the-alexa-skills-kit.html, last accessed 2019/10/6.

[22] Czrabode.: Creating Complex Automation Using IFTTT and Stringify, https://medium.com/@czrabode/creating-complex-automation-using-ifttt-and-stringify-72ec01df60e0 (November 2017), last accessed 2019/10/7.

[23] What is Vuex?, https://vuex.vuejs.org, last accessed 2019/10/8.