CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Department of Control Engineering

Design of Smart home Multi-layer Control system

Diploma thesis

Jan Dolezal 2008

Design of Smart home Multi-layer Control system

Abstract. This thesis proposes an optimal structure of a control system suitable for smart homes. There are many complex and general systems for automation, but the same system used for factory automation is not suitable for application in an apartment. Author takes technical, economical and especially user points of view into account to design an innovative three-layer control system. It meets requirements of various kinds of users better than the currently used two-layer systems. A detailed description of the new proposed middle layer is provided, including connection to existing systems, innovative user interface and hardware implementation. Simplified visual logic system is designed so that even an unskilled user can use middle layer to perform basic maintenance and modifications by himself without risking damage to the bottom-layer functions. Solutions optimized for home automation could finally cover gap at the market and, thus, intelligent installations could become frequent at homes.

Abstrakt. Tato diplomová práce navrhuje optimální strukturu řídicího systému vhodného pro aplikaci v inteligentních budovách typu bytu či rodinného domu. Řešení používaná v průmyslu často nejsou vhodná pro domovní aplikaci. Autor bere v úvahu technické, ekonomické a zejména uživatelské faktory, aby navrhl inovativní trojvrstvý řídicí systém, který vyhovuje požadavkům různorodých typů uživatelů lépe, než současné dvouvrstvé systémy. Práce poskytuje detailní popis nové střední vrstvy, včetně propojení s existujícími systémy, inovativní uživatelské rozhraní i vhodnou hardwarovou implementaci. Zjednodušený systém logiky je navržen, aby i uživatel bez znalostí řídicích systémů mohl díky střední vrstvě sám provádět základní údržbu a změny systému, aniž by riskoval poškození vitálních funkcí. Řešení optimalizovaná pro domovní automatizaci tak mohou konečně pokrýt mezeru na trhu a inteligentní instalace se stát běžnou součástí domácností.

For a 2-page introduction of the thesis, please refer to Appendix B

Table of contents

List of figures

List of acronyms

BACS	Building Automation and Control System
CCTV	Closed-circuit television
DDC	Direct Digital Control
DOM	Document Object Model defined by www.w3.org
FIFO	First In First Out
GUI	Graphical User Interface
HAI	Home Automation Inc.
HAL	Home Automated Living
HVAC	Heating, Ventilating, and Air Conditioning
LAN	Local Area Network
.NET	computing platform developed by Microsoft
OPC	originally OLE for Process Control, OPC Foundation is organization for standards in automation
OPC XML/DA	OPC eXtensible Markup Language / Data Access
PCI	Peripheral Component Interconnect
PDA	Personal Digital Assistant (electronic handheld information device)
PLC	Programmable Logic Controller
SCADA	Supervisory Control And Data Acquisition
SoftPLC	computer acting like Programmable Logic Controller via software
UPB	Universal Powerline Bus
WAN	Wake on LAN
WPF	Windows Presentation Foundation
X10	manufacturer of home automation hardware www.x10.com
XAML	eXtensible Application Markup Language
XBAP	XAML Browser Application
XML	eXtensible Markup Language

Preface

The thesis is based on the research carried out at the Faculty of Electrical Engineering at Czech Technical University in Prague. The governing thoughts of this thesis including three-layer concept, middle layer definition, simplified logic, hardware structure and guidelines for centralized control are my own ideas. GUI design results from the middle layer definition, but is also inspired by many other control systems that I studied – see Appendix A.

I wish to express my sincere gratitude to my supervisor, Pavel Burget, for all of his assistance and guidance over the past few years. I especially treasure the experience gained in CEPOT projects Mr.Burget initiated and thank him for the opportunity to write my thesis abroad.

I would also like to thank Jan Vidim from Domat Control Systems and Marek Kulvejt and the rest of Energocentrum for consultations and practical experience.

Smart homes were already my motivation to enroll at Faculty of Electrical Engineering during the last year at high-school. I am glad that my interest lasted till graduation and I can hereby present the fruit of my work.

If only some ideas given in this thesis could be put into practice and contribute to improvements in living standard of the people on our planet - that would be the greatest satisfaction for me and make my work meaningful.

1. Introduction

Home Automation is the integration and control of lighting, security, multimedia, climate control and other electronic systems within a household. It is a subset of building automation with particular focus on living spaces in a scale of an apartment or a family house. The main purpose is to make everyday life more comfortable, safe and energy efficient.

Devices in a Smart home are interconnected into a network and controlled by specialized software called Control system. It connects to the devices, monitors their state and reacts to events. The control system performs many tasks associated with the household instead of the residents, enabling them to spend more time on matters that are really important. The control system not only enables comfortable light scenes, daily programs or remote access. It also takes care of optimal heating and ventilation to keep the air fresh and temperature comfortable, but to safe power when nobody is present.

For example, y*ou can switch dinner light scene to a scene suitable for watching movie with a remote control, or lock downstairs door and turn off all the lights by a single button in your bedroom. You wake up gently to the melody of your favorite tunes, having boiling water for the coffee ready and garage heating warming up your car. When you leave, all the lights and appliances turn off by a single click, the door locks automatically and security system activates. Smart home can run lawn sprinklers according to weather forecast from the Internet, deliver new digital photographs of your friends to the LCD picture frame or warn you when the milk expires.*

Even though the idea sounds very technical and futuristic, a true Smart home is intuitive and simple to use so that everyone in the family enjoys living there. Home automation can simplify our living in the same way automatic washing machine simplifies washing the clothes. In the future, Smart homes are likely to become as common as washing machines. Currently, every second new house in the USA has some kind of structured installation and mass production of smart homes has started in Western Europe. With a majority of mankind living in the region, it is also Asia that calls loud in demand for energy efficient and comfortable solutions in the massive construction works that take place over there.

Research of home automation thus gains significant importance. But until recently, it has been neglected in the shadow of industrial automation and building automation focusing on office spaces. By now, the market of smart homes has grown big enough to have specialized solutions. Plentiful small home automation providers emerged and also the automation giants started to realize the need of solutions tailored for home usage. However, offered systems are often expensive, complicated and inflexible. On the other hand, really easy and affordable solutions such as X10 still have very limited functionality. That’s why home automation market falls far behind the expectations and possibilities. We already have all the technology and hardware to build commercially successful Smart homes that today people only dream of. It is all just the matter of good software.

An optimal control system for home automation will be proposed in this thesis using the top-down approach. Driven by global economical trends and user demand, we will first define the user interface independent on underlying system. Later we will look for suitable hardware installations.

2. Problem overview

Control software is both the brain and the heart of a smart home, which determines the functionality, user-friendliness, stability, efficiency and therefore success of home automation. In this chapter, we will define control system layers, look at current control systems used for home automation, analyze their problems and finally understand how three-layer concept can solve them.

2.1. Traditional two-layer control system

Majority of current control systems for home automation consist of two layers, which we can refer to as a *traditional concept*.

2.1.1. Bottom Layer

The bottom layer connects to the physical process, defines and organizes data points and implements the logic. It is the core of a control system, where control algorithms are executed. It is often a sophisticated application that can be used for a wide range of automation tasks from sorting objects according to color to controlling a nuclear power plant. A skilled programmer who is both experienced in programming that particular environment and understands the controlled technology is needed to configure this layer.

In most of the systems, there is a PC based configuration tool for bottom layer program which runs on a devoted hardware such as a PLC or directly within the units in distributed systems.

2.1.2. Top Layer

The main task of the top layer is usually visualization and control over the process, such as modifying values and executing predefined actions. This layer is generally referred to as *SCADA (Supervisory Control And Data Acquisition)* which indicates that it is typically used in the supervisors' control room. SCADA often has graphical interface and is easy to use. In smart homes, top layer application often employs touch panels.

2.1.3. Problems with two-layers for home automation

The concept of two-layer systems originated from industrial environment and therefore fits needs of a factory, but proves unsuitable for home automation.

	Industry	Smart Home
Speed	**critical**	not critical
Stability	**critical**	some tolerance
Users	trained	**untrained**
Price	not most important	**essential**

Table 1 - Difference of requirements for Industrial and Smart Home control system

Increased speed and stability is always on the account of higher price of the technology. Because in industry, speed and stability are critical, the same system will likely be too expensive for home usage. This does not mean that home automation system can be very slow and unstable, but the optimal balance between performance and price is very different for both application fields. We can tolerate an occasional 100ms delay between pushing a button and switching the light at home, but far less than 100ms response must usually be guaranteed in the industry. Current high price is the main reason why majority of homes are still not intelligent.

Users

Traditional control systems were designed for industrial users. On lower level, it is a well trained system expert and programmer, who implements and updates the all the control algorithms and designs SCADA visualization. A trained supervisor uses top layer to view important information about the system, check alarms, set variable values and run predefined actions. In contrary, a typical home-automation end user knows nothing about control algorithms, programming and haven't received any training.

***Example**: A woman with her family lives in a smart home with a two-layer control system. One day she buys a new desk-lamp and connects the power. She already got used to the great "Leave the house button" that automatically turns all the lights off. It saves so much time and running around the house to check everything when she leaves. Now she has to come back, because the new desk-lamp has not turned off. How could it, the control system still do not know about it! She asks her son, who is quite skilled with computers, to connect the lamp. He first searches the touch screen menu, but no option to add a new device. Then he remembers how the system integrator used some complicated software as he asked them which light they want to control by which switch, when he was configuring the system. Son succeeds in executing this software, but is scared off by lines of programming code and complex schemes with many symbols that he is not really sure about. He doesn't want to spend hours reading a manual and still risk that he will mess all the system up. Now they have to make a decision whether just leave the lamp uncontrolled and always turn it off manually or call the integrator to come and set it up – which would cost much more than the lamp itself.*

General versus specialized solution

Some providers develop special software for home automation, while others prefer to use a general control system. Both approaches have their pros and cons.

It is easier and more economical to develop one general control system environment with lots of available features than a specialized system for each application. We can surely perform all the tasks of home automation with general and customizable software that can control a whole factory. But the generality necessarily results in a more complicated system that becomes too difficult to understand for the user. *How many people prefer a highly sophisticated remote control with 50 customizable buttons and scripting language capabilities, over an intuitive remote control that is designed especially for what they need to control? Why Apple's simple control interfaces have became so successful?*

Development of a specialized control system requires much additional effort and the system loses possible functionality with every simplification. Should we implement a function that is only required by a few users, but will make the system more complicated for the rest?

Most of the current specialized home automation software is either very easy to use, but also very limited, or configurable with scripting support and complicated.

2.1.4. Possible solutions

Taking those problems into account, we have couple possible ways to adjust the control system to suit the requirements of home automation.

One option is to create different access rights for programmer of the bottom layer and end user who only wants to make small modifications. We could lock the critical control loops and even hide sophisticated settings for the end user so that he can use all the power of bottom layer to implement own ideas without the risk of damaging the critical functions. We can also add a library of function specific for home automation. A problem stays the complicated programming environment as a barrier for most of the end-users.

Another possibility is to add more functions to the top layer SCADA, so that the user can add and configure devices or even change the logic. This leads to a rather complicated SCADA which would not be intuitive for many unskilled end-users. The top layer's input and screen view is usually limited and it could be difficult to display all the information necessary for system configuration on a small touch screen or a television screen. The large screen, keyboard and mouse of a PC provide much more efficient input, than touch screen or a TV remote control. An ordinary PC on the other hand is not suitable as top layer control for users to apply a light scene or turn the heating up.

Both stated solutions result in one layer trying to cover a wide spectrum of functions for different kinds of users, which leads to big compromises and as a result, the user interface is not really suitable for anyone. Therefore, adding one layer to the control system seems as an appropriate solution. This middle layer would implement special interface to cover the specific needs of home automation and its users.

2.2. Proposed three-layer control system structure

2.2.1. Multi-layer control systems

Multi-layer control systems have already been successfully used in other application fields, such as traffic control (1). However, three-layer architecture used in robot control system (2) (3) refers to a completely different idea of layered system[1].

Even though the term "three-layer" may sound familiar from other applications, the control system structure described in this thesis in my own idea and I haven't found any existing control system that would already employ this kind of architecture.

The one layer I add to a traditional concept can be seen as a middle layer in the proposed system. Designed functionality and interface of each layer is based on the user type it is supposed to serve. While top layer must be easily operated by anyone, middle layer requires advanced computer knowledge and bottom layer is designed for automation expert.

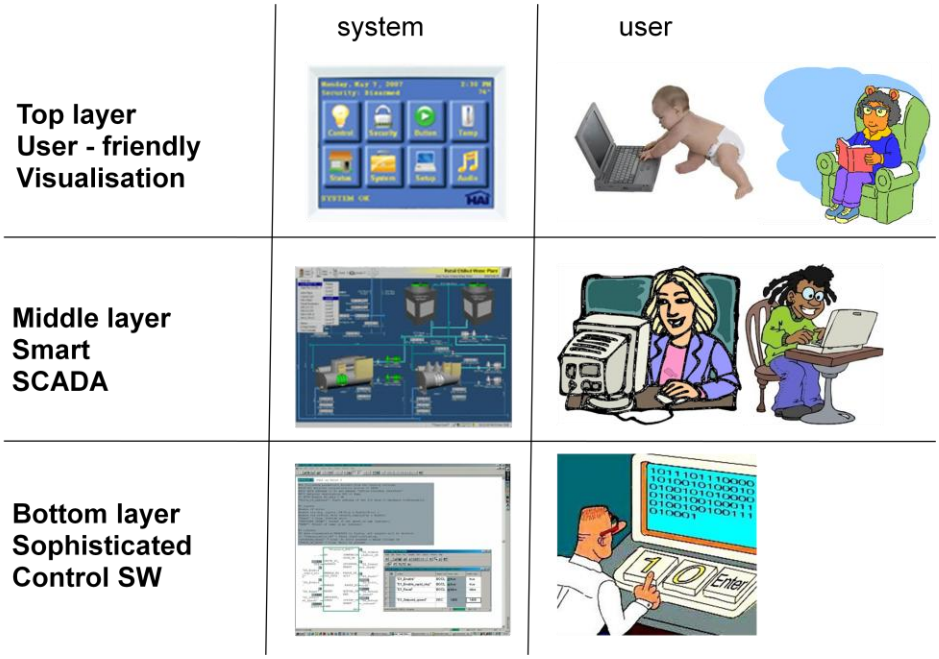


Figure 1 - User structure of three layer system

Let us declare the specification of new layers compared to the traditional concept.

[1] In robotics three layers refer to three separate computational processes: a reactive feedback control mechanism, a reactive plan execution mechanism, and a mechanism for performing time-consuming deliberative computations.

2.2.2. Bottom Layer

The same application that serves as the bottom layer in the traditional concept can be used as a bottom layer in the new concept with rather minor modifications needed for communication with the middle layer. It is a robust control system core, which can be used generally for any kind of system including factory or home automation. Many software packages of this kind already exist, typically developed by big control hardware manufacturers. Some of those systems are not completely open for other competitions products, so we can also consider using a third-party general control system that supports wide variety of platforms.

Producer	Control System SW
Siemens	Simatic Step 7
B&R	Automation studio
Domat	RcWare
Teco	Mosaic

Table 2 - Examples of general Bottom layer Control SW. A list of specialized home automation SW can be found in Appendix A – Overview of specialized home automation SW

All the communication with the physical system is managed in this layer and rather abstract data structures are presented to upper layers.

A typical user of this layer is a programmer and technology expert in one. In case of home automation, it is a well-trained integrator from the company, which implements control systems. He is needed mostly during the installation phase. He connects control system to the physical system, defines data points and implements main control loops, particularly HVAC. Unlike in the case of the traditional concept, he is not needed for small logic and structure modifications, because those can be performed in the middle layer.

2.2.3. Top Layer

Even the top layer application can be essentially the same in the three-layer system as the top layer in a traditional two-layer system. However, because the more complicated tasks are moved to the middle layer, the top layer can be even simpler and therefore easier to use.

User interacts with the system mainly through the top layer devices, which include:

- Switches
- Remote control
- Touch Panel
- PDA / Smartphone
- Media Center[2]

[2] A great example of intuitive top layer interface for Media Center is Home Control by Home Automation, Inc. http://www.homeauto.com/Products/Software/MCE.asp

Figure 2 - Windows Media Center

Figure 3 - Linux Media Center MAC OS Media Center

Unlike in the lower levels, there is generally **no authentication required in the top layer** for three reasons:

- Top layer devices are generally on the same level as a classic switch that turns on a light. As well as any guest can freely turn light when present in our house, anyone can also use the Top-layer applications to control supported devices.
- The top layer only exposes safe actions so that no damage to the system can be made by its misuse.
- Top layer control is frequently used and authentication would inadequately slow down the control.

The top layer control must be very intuitive so that even the least computer-literate end-users can use it easily. In home automation those users include little children and elderly people.

2.2.4. Middle Layer

Middle layer makes the main difference between traditional and suggested three-layer concept. The main purpose of this layer is to enable advanced computer-literate end-users make basic modifications to the system without facing problems caused by editing the bottom layer.

Middle layer is essentially simplified bottom layer software, which is streamlined for home automation. It connects to abstract data points and functions provided by the bottom layer, organizes them into a logical structure and connects them with simple logic. A typical task performed within the middle layer is to change light scene, time schedule settings or connect new device to the system.

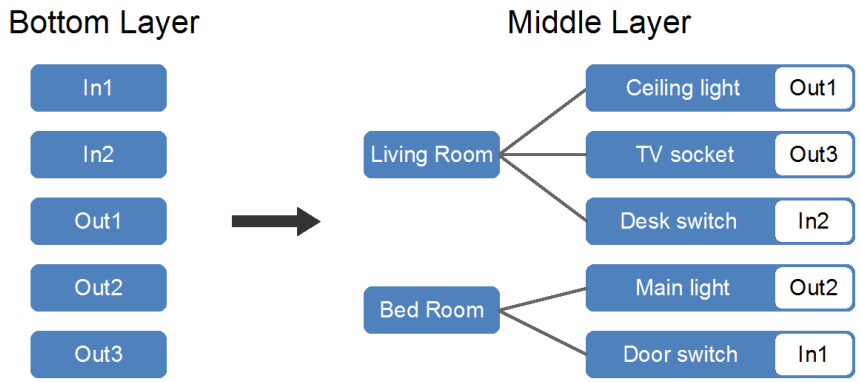


Figure 4 - Middle layer organizes bottom layer data points

Middle layer is intended to be used by slightly advanced computer users. Working with middle layer should be about the difficulty of making a PowerPoint presentation. It is clear, that there is not such a skilled user in every household, but almost everybody has people of this ability amongst neighbors or friends. For example an old grandma living in an apartment alone can ask her middle-school grandson to help her make modifications in the middle layer. It is possible to connect remotely via secured internet connection, so finding assistance is not difficult. It is definitely a great option in addition to calling an expensive programmer to make changes in the bottom layer of a traditional system.

The main reasons for introducing middle layer:

- Three layers reflect the user structure much better than two-layers.
- Middle layer gives the end-users a relatively safe opportunity to perform basic maintenance tasks by themselves. In traditional systems, the end-user often has to call the bottom-layer integrator to perform relatively simple tasks. This leads to user discomfort, increased costs and overwhelming demand on the programmers. Shifting basic maintenance to the end-user is essential for mass-installation of smart homes. *For example, hardly anyone calls technician to tune channel on a TV or install a new printer to the computer.*
- Middle layer solves the problem of compromise between variability of general control system and functionality of a specialized one. Bottom layer can therefore stay a general control system without little need for special modification for home automation. On the other hand, the middle layer is entirely designed for home automation and therefore can be more efficient and easier to use. We can keep middle layer simple, because we always have a back-up option to implement non-standard functionality in the bottom layer and satisfy all the customers, which is a great advantage over current simplified end-user oriented home automation control SW.
- Separation of critical control loops and user-defined logic is safer.
- By shifting some advanced functionality from top to the middle layer, the top layer SCADA can be even simpler and easier to use for the unskilled users.
- A layered structure allows higher abstraction similarly to the typical 7-layer network architecture. Multi-layer systems are generally more robust and flexible.

We will look at the middle layer in detail in the next chapter.

3. Middle layer specification

In this chapter, we will learn what the middle layer is and how it works. We will define data structures, suggest a simplified logic system and its rules and touch the topic of alarm management. The key point is the user interface, which demonstrates the main features of middle layer. At the end of this chapter, we will discuss connection with other layers and priorities.

Middle layer application is primarily designed for an ordinary PC. The actual hardware, where middle layer application runs, depends on particular system architecture. The ideal case is a control PC / SoftPLC, which runs at all the times and hosts both bottom layer and middle layer software. This is a robust low-maintenance machine typically placed in a utility room. We can take advantage of operating system[3] that allows multiple partitions on one CPU and therefore guarantees higher stability. Middle layer can be accessed via remote interface from any computer connected to the network / Internet. However, most of the current systems do not require a computer based machine running at all the times. Instead they use PC for configuration only and the system behavior is then downloaded into specialized hardware. In the latter case, middle layer can be used just as the bottom layer software for configuration only. This solution is generally more stable, but has limited functionality.

3.1. Data structures

Carefully designed data structures are necessary in order to represent the smart home in a complex, yet not too complicated fashion.

3.1.1. Data types

For correct interpretation of the data, we define a set of data types. Since the middle layer manages abstract information independent on other layer's actual implementation, but has to be compatible with all possible implementations, those data types should be generally a flexible superset of commonly used data types. *This means, that for example integer data type has the optional parameter "maximal value", but this parameter does not have to be assigned.*

We can represent most data using this set of data types:

- **Boolean**
  By default, "0" can also be referred as "false" and "1" as "true". We can also redefine those strings to represent the real situation better. Therefore, for some Booleans we can set "0" as "off" and "1" as "on", or "no" and "yes". This option can make the data more understandable for the end-user, but we should limit to cases when the meaning is really evident. For a couple such as "left" and "right" we'd rather use a List.
- **Integer**
  Different systems use different Integer precision. In order to maintain the precision for all of them, we have to choose the highest one.

[3] An example of such operating system is SYSGO (www.sysgo.com)

Optional parameters include:

- Unsigned
- Minimal Value
- Maximal Value
- Increment

- **Real**

  This format is used to represent any number. High precision has to be used to represent both float and double formats well. The application must also be careful about real overflow / underflow caused by exceeding precision limit. Such violations should be reported, but functionality preserved. A possibility of maintaining some functionality during overflows is rounding to the precision limit.

  Optional parameters:

  - Minimal Value
  - Maximal Value
  - Step (indicates maximal precision-rounding while reading the variable)
  - Increment (never smaller than step)

  The main difference between Step and Increment is that Step is used for reading, while increment for setting the variable. If Step of temperature is set to 0.1°C, than the real temperature value of 20.73195°C appears as 20.7°C. Even though the temperature preset for A/C can be modified by 0.1°C, we might want to set the Increment to 0.5°C so that the user can set the temperature faster using up/down control.

- **String**

  A dynamic implementation is suitable to avoid length limits.

- **List**

  A finite ordered list of choices.

  Loop parameter indicates whether the first item also follows the last one.

- **Timestamp**

  Specifies exact date and time

- **Percentage**

  This is not as common data type as the ones above, but Percentage is a very convenient way of expressing a relative number – *for example the light is turned to 30%, laundry machine progress 80% finished and so on.*

  Percentage is basically a Real value, but we have to be careful about the conversion. In middle layer, the conversion between Percentage and Real or Integer is very natural if Minimal / Maximal Values are defined:

$$Percentage = 100\% \times \frac{Current\ Value\ - Min\ Value}{Max\ Value - Min\ Value}$$

On the other hand, for mapping between layers we often consider 0=0% and 1=100%.

$$Percentage = 100\% \times Real$$

3.1.2. Data points

Representation of a system variable can be called a data point. There are different data point structures for different data types.

Each data point can have a number of properties:

- **Name:** serves for data point identification. Full name must be unique within the whole application, which usually results in repeating area in all the names such as *"Living Room Temperature", "Bedroom Temperature"*
  Another option would be to enable name unique within the area, so that with area prefix it is unique in the whole application. The prefix can be invisible to the user and names therefore shorter. In the second case, however, the area must be always displayed with name to avoid confusion.
- **Data Type**
- **Value:** of the specified Data Type
- **Data mapping:** Specifies particular connection driver and variable name in other layer
- **Description**: A brief explanation of what the data point actually represents, such as "Temperature in the living room"
- **Permissions:** access information stating which users can read or modify the Value

Other properties are dependent on the Data type. They include for example:

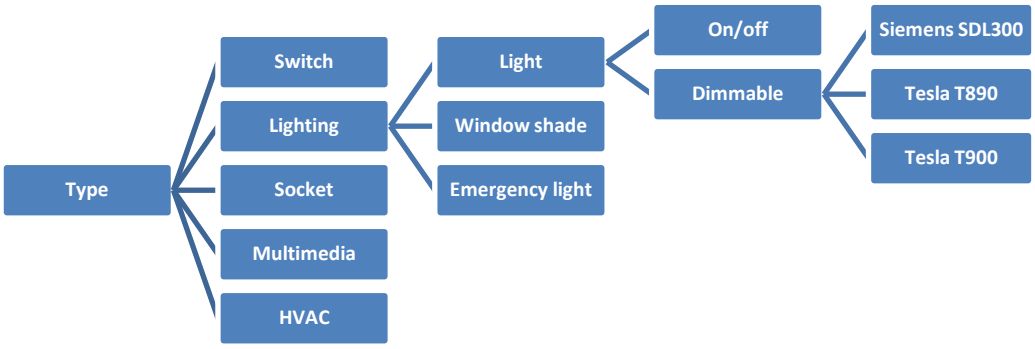
- **Min/max value**
- **Increment**
- **Step**
- **Timestamp**: time of measurement or last change of the value
- **Min/max alarm level**: exceeding this value causes alarm
- others…

In bottom layer, all the data points are on the same level, without much organization. In case some bottom-layer application supports organization of data points, we will not use it. It is up to the middle layer to organize data-points according to their type and physical location, which will be very useful later in the logic.

We will add following optional parameters to all the data points:

- **Type**

  Type specifies what kind of object the data point represents. All types are organized in a customizable hierarchical tree[4], such as:



  We can assign a general object type "Light", but if we want to be more precise we can specify "Dimmable". We can also assign the actual model if its definition library is present. Even then the object is still member of all "Lightning", "Light" and "Dimmable" types.

- **Area**

  Area parameter indicates the real physical position of the object. The list of areas is customizable and reflects actual home setting such as: *Hall, Living room, Kitchen, Bathroom, Bed room, Children's room, Patio, Garage, Garden*

Defining those attributes allows us to perform powerful global action with quantifier like "*turn off all the appliances and lights in the kitchen*". It can also be used for filtering data points or rules associated with one room like "*view all the logic rules connected with lighting*" and much more.

In middle layer, we will require more functionality from the data points than just storing data. To avoid timing conflicts in logic, each data point is equipped with its own timer. Each data point also tracks own history so that it can raise alarm when a conflict is detected – i.e. one rule systematically turns light on, while another one turns it off.

This functionality can be implemented centrally, or in a distributed manner. In the later case, we could regard the middle-layer data points as agents.

3.1.3. Variables

In addition, to data points linked to data outside of the middle-layer, we can also define internal data points that are used only within the middle layer, but can have the same types and properties as any other data points. We will refer to any kind of data point simple as a variable.

[4] Hierarchical tree structures idea as a general data type comes from ENERGOCENTRUM PLUS, Ltd.

3.2. Application structure

Before we start describing individual components, let's have a look at the structure of the whole control system to get the big picture:

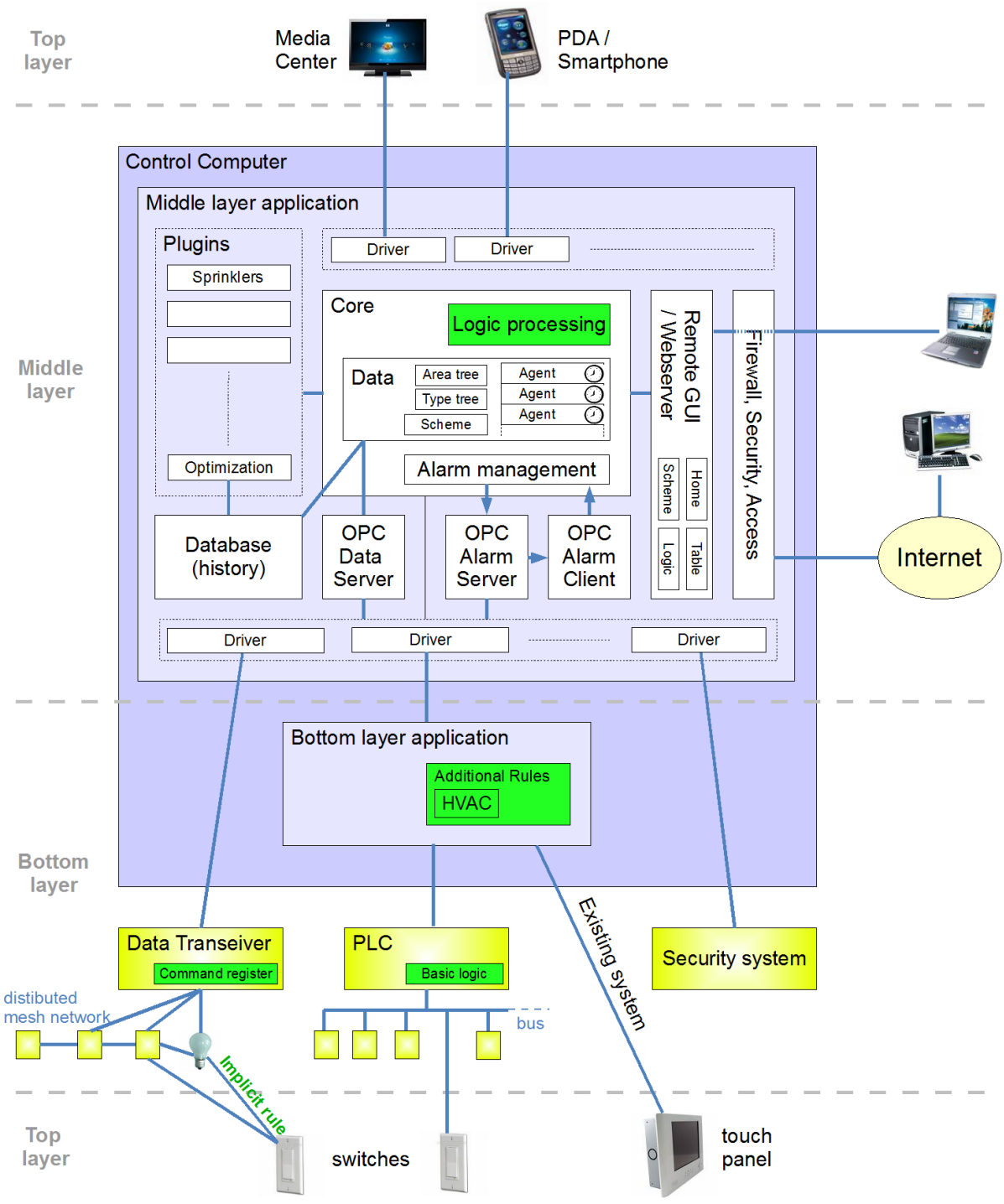


Figure 5 - Application structure

Figure 5 shows the application structure with main building blocks and important (but not all) data connections.

- **Core**: In the core of the application agents representing variables are stored, logic executed and alarms managed.
- **Alarms**: Generated alarms are published on the server to synchronize them with other applications. Selected alarms are processed with the Alarm client.
- **Remote GUI**: GUI is accessed remotely via secured connection. It can be for example web-server generating XBAP (*XAML Browser Application*).
- **Security**: The application downloads information such as weather forecast, device drivers, plugins and updates from the Internet, but also exposes its data, so firewall, encryption and other security measures are used to prevent virus infection, or unauthorized access.
- **Data**: Actual data are shared on OPC server and historical data archived in a database.
- **Plugins**: Plugins can add additional features or support for specific hardware present in the system.
- **Drivers**: Communication drivers enable connection with other Top layer and Bottom layer systems.

We will discuss some elements shown at Figure 5 in detail later within the text. Please note that there can be **multiple bottom layer and top layer** applications present, but **only one middle layer** serving for their connection.

3.2.1. Agents

Each variable is represented by a complex object that can be called an agent. In addition to storing the actual data, the agent includes a timer and monitors data history to indentify conflicts and cycles in logic rules.

Timer

Timers that affect the variable are transferred directly to the variable agent, rather than staying in the logic or elsewhere outside. When a new timed function or a simple value change is assigned to the variable, it replaces the old timer to avoid conflicts.

For example a rule is defined that will turn on light on the porch when a movement in the garden is detected and turn it off after a given period of time. What happens in an ordinary system if the user during this time comes from garden turns the porch light off and then returns back to turn the light on and read a book on the porch? Suddenly the light goes off, because the remaining timer connected with the rule in logic sent command to turn the light off after given period has elapsed.

This problem is solved in an elegant way when using the agent. The rule will simply assign the variable timer a plan to turn on immediately and turn off after X minutes. When agent receives user's command to turn the light off, it simply annuls the timer. This way we ensure that only the last command is active and avoid conflicts.

History monitoring

Agent keeps track of the several last changes that occurred with the variable in a FIFO memory. It monitors value, exact time of change and ID of the object/rule which caused the change. Agent looks for a repetitive pattern in history that appeared within a relatively short time frame. In case this occurs, it fires alarm which includes both senders IDs. Thus it is easy to identify what causes the conflict - and if it is desired behavior, to ban this kind of alarm from reoccurring and bothering in the future.

Example:

Timestamp	Value	Sender
11/29/2007 14:05:50	0	Rule ID 346
11/29/2007 15:30:12	1	Rule ID 202
11/29/2007 15:30:12	0	Rule ID 543
11/29/2007 15:30:12	1	Rule ID 202
11/29/2007 15:30:12	0	Rule ID 543
11/29/2007 15:30:13	1	Rule ID 202
11/29/2007 15:30:13	0	Rule ID 543

Table 3 - Example of history monitored by an agent

The history track displayed at Table 3 reveals that there are two rules, which are both attempting to set the variable, so an alarm is fired.

The number of occurrences we will watch for should be small enough to fit within the FIFO memory and react quickly, but large enough to avoid firing alarm in situations like: *the user accidently clicks a button twice which turns a device on and back off, so that the user clicks the button again to turn it on.* About 3 to 4 occurrences is a reasonable number to look for.

The time frame within which we look for conflict should be long enough to catch even conflict involving user's reaction (for example user is setting temperature up, but some rule is systematically pushing it down), but short enough not to include cases caused by regular actions, such as turning bathroom light on and off many times during the whole day. A reasonable time frame would therefore be about a minute.

3.3. Simple logic

Because we want to allow the user to define basic logic control by himself, we need to design a new concept of logic, which is as simple as possible, yet allow high functionality. There is always a tradeoff between those two qualities. In the bottom layer, we require maximal functionality which makes the system of logic too difficult. Here comes the advantage of three-layer application. Knowing that we already have the maximal functionality in bottom layer (that can be used in case we have some uncommon request), we can design much simpler and therefore user-friendly logic in the middle layer.

Our goal is to develop an intuitive and user-friendly system of logic for ordinary advanced computer user without knowledge of control systems. It is required mostly for *switch logic*, which means logic rules associated mainly with turning appliances on and off, both according to pressed buttons and time schedules. A typical application is programming scenes or a *goodnight button*[5]. It is not meant for HVAC or other complicated control.

3.3.1. Current logic systems

Let's first look at current systems of logic and then their possible simplification. While general automation software usually features rich logic with many operations and functions, programs specialized for home automation typically allow simple text based programming, sometimes reduced to a list of if…then statements. To cater for more demanding users, some home automation software also allows scripting using standard programming languages.

Most of the systems of logic originate from PLC programming languages. There are five control system languages defined by IEC 61131-3 standard (4):

- **Ladder diagram** (LD, LAD) – represents simple graphical sequential control originally invented to describe logic made from relays. It is suitable mostly for binary values.
- **Function block diagram** (FBD) - describes a function between input variables and output variables. Blocks can contain even complex functions. The links are oriented and can represent any kind of information.
- **Structured text** (ST)
- **Instruction list** (IL) – includes Statement List (STL)
- **Sequential function chart** (SFC) – graphical language based on GRAFCET used to program processes that can be split into steps

Graphical control is definitely more intuitive than textual[6] and home automation is too complex to be described by a state automat. Therefore we can get inspired by LAD and FBD forms.

[5] So called „Goodnight button" is usually placed in the bedroom. It launches automated series of actions that the user would normally do before going to sleep, such as turn off the devices and lights, lock the house, slowly dim light or fading music in the bedroom and enter the "night mode" in which lights are by default turned on to about 70% of their capacity and with slower transition, to make it more comfortable for accommodating sleepy eyes.

[6] An example of a textual event based control system for home automation is JDS Event Manager (16). Such logic typically involves a simple programming using if…than…else statements, which is straight-forward for any programmer, but not intuitive for amateurs.

An excessive number of function blocks are necessary in order to achieve full control possibilities:

- **Basic logic operations**: AND, NAND, OR, XOR, NOR, NOT
- **Mathematical functions**: Min, Max, Equal, Abs, Signum…
- **Switches**: Switch, Multiplexer…
- **Counters**
- **Timing**: On/Off-Delay, Retentive On-Delay, Wiping relay…
- **Flip-flop circuit**: RS, T, D, JK…
- **Signal generators**
- **Time programs**: Monthly, Weekly, Daily plan…
- **Others**

We have two ways of making the user experience more intuitive and we will employ both of them:

- Simplify the logic itself
- Simplify the user interface

3.3.2. Simplifying the logic

Since our main focus in the middle layer is the logic control, we can restrict Real signals only to the very input and output parts of our logic rules and only proceed with Boolean variables through the logic. This restricts the functionality, but reduces the number of blocks immensely and limits conflicts resulting from incompatible data types. What is important, this simplification does not interfere with our needs.

Some functions may be appreciated by a programmer, but confuse the end user. In the middle layer, we will only use necessary functions and those that can be easily understood.

For example NAND and NOR functions will find their use in the bottom layer, but we can replace them by simple NOT, AND, OR in the middle layer.

3.3.3. Simplifying the user interface

Shapes

Lots of difficulties and misunderstandings stem from using various incompatible signals/variables within one scheme. There is always a Boolean signal, and one or more types of analog/digital signals which are not compatible. Every pin at a function block can only connect to one signal type. For some signals, there are also specific function blocks.

For example, we cannot connect a Real value to "AND" block or compare Real and Boolean values in a comparator block.

We should make it very clear what kind of signal is to be used at which place. In some control systems, this is achieved by different style of the connecting line, in some control systems, there is no visual difference between logic and analog values. We can use the shape of the function blocks itself to

provide intuitive type information to the user. For example, we can associate logic values with a cornered shape and analog values with round shape.

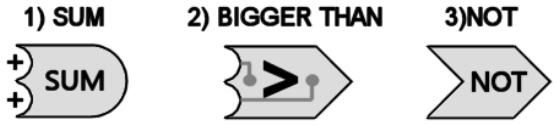


Figure 6 - Function blocks for analog and logic values

We can then easily identify that the SUM function block at Figure 6 takes two analog values as input and outputs also analog value. On the other hand, the second function block (BIGGER THAN) has two analog values as input, but results in a logic value. Finally the NOT function takes one logic input and has one logic output. The user can very intuitively distinguish that it is possible to connect output of the SUM as one input of BIGGER THAN or connect NOT block after the result of BIGGER THAN. It is apparent that the result of SUM does not fit in the negation block. We can also feel that none of the three blocks can be nor first nor last function block in the rule, because of its edges. Variables have flat left side so that it is intuitively placed at the beginning of the rule, while actions right side is flat to close the rule.

Colors

We will also use colors to indicate state of function blocks and devices. Neutral gray is used when we view the logic or scheme in offline mode or for variable types where it is difficult to distinguish whether the value is high or low. For binary and percentage values (and possibly real values with min/max value attribute), we can indicate the low state (i.e. False, 0, 0%, OFF, NO, passive...) by a cold color and high state (i.e. True, 1, 100%, ON, YES, active...) with a warm color. In this thesis I chose blue and yellow, but it could be red and green or other combinations.

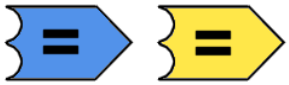


Figure 7 - Color distinguishes which comparator results in false(blue) and which true (yellow)

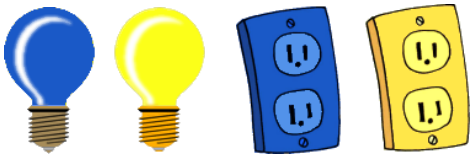


Figure 8 - The same colors used for scheme gadgets - off (blue) and on (yellow)

We could go even one step further and show intensity of real values and percentages by a scale of colors (i.e. light color for small numbers and darker color for higher number)

3.3.4. Rules[7]

We can define the logic as an ordered list of rules that are processed in order similarly to the ladder diagram at PLC. The order therefore defines priority of processing the rules. First rules are fired sooner, but their results can be overwritten by following rules, which throws a low-priority alarm warning. Unlike a typical PLC program, most of the rules in switch logic are event-driven and independent, so their order will not matter.

A rule consists of three parts:

- **Event / Test:** Event is typically fired by change of some *Value*, while Test monitors the Value continuously. *For example, „Button pressed shortly" is an Event, but „Temperature smaller than 5°C" is a Test.* Both Event and Test result in a Boolean value. Event can be also fired when SMS or email are received and by voice recognition system if voice command is recognized.
- **Logic:** A simple Boolean logic combines result of Events and Tests from the previous part. *For example, the logic could be "if Event1 and (Test1 or not Test2)".* If the result of the Logic is true, the following Actions will be fired.
- **Action:** Basic action defines a list of variables and instructions how to change them and when. Action can also raise an Event or send email…

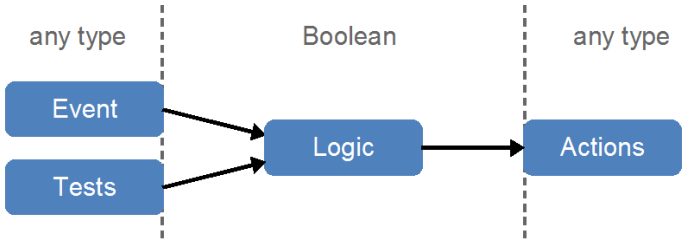


Figure 9 – A rule

Events are used to handle inputs and Actions for outputs. Even though real variables come to the first part of the rule (Event / Test), they do never enter the binary logic. Real variables do not propagate through the logic directly, but can again be accessed in the Actions. This separation of rule into three parts makes it obvious what kind of signal should be used at every place within the rule and reduces the number of available function blocks and therefore simplifies the interface.

[7] Actions or the whole Rules are sometimes referred to as "macros" in other software, but here we will reserve the word "macro" for more complicated scripting, rather than simple condition based rules.

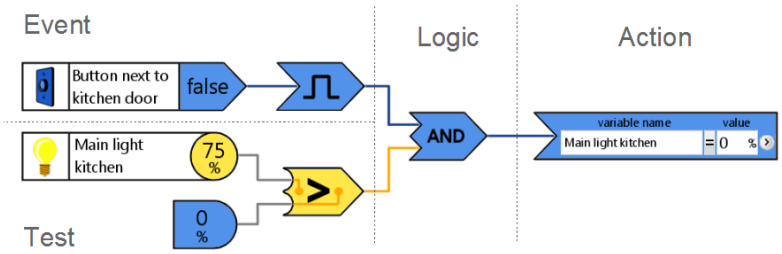


Figure 10 - Example of a rule

Value

In the place of a Value before Test or Event can actually be one of the following:

- **Variable** (Data Point or User defined Variable)
  The left side of examples at Figure 11 is flat, signifying that Variable is placed at the beginning (left side) of the rule. If the Variable is represented by gadget in the scheme, the corresponding icon is shown on the left side.
  Variable name occupies middle.
  The round element on right side suggests it is analog value, cornered shape is again used for logic value. If the actual value is available, it is shown together with unit in the circle, which changes color according to the value.

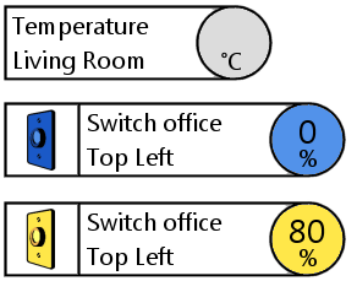


Figure 11 - Examples of Variable function blocks (not available, low, high states)

- **Constant**
  Constant can be of the same types as variable, analog or logic.

Figure 12 - Examples of Constant function blocks (3 analog and 2 logic Constants)

- **Time program**
  We can create yearly, monthly, weekly or daily time programs, both analog and logic. The output of a time program is a variable of the given type. It can be used in exactly the same way as other Variables or Constants.

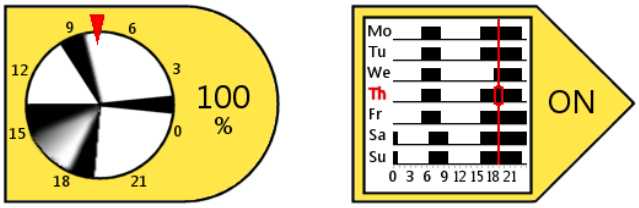


Figure 13 - Examples of analog daily and logic weekly Time programs

Please note, that Time program is essentially a periodically changing input variable, while Agent's Timer is rather a timed list of changes to be applied to an output Variable.

- **Analog function**
  Analog function is a function block that outputs a Value and itself can take Values as inputs. The basic analog functions include sum, minimum and maximum. Each of those functions can take two or more inputs.

Figure 14 - Examples of Analog functions

Event

An Event is a logical zero all the time except for the moment (one cycle) when a watched event occurs. Event can originate in two ways:

- **User-defined event** with unique name as a result of an action. It can also be allowed to be fired remotely. *For example "Leaving the house" or "Car arrives".*

Figure 15 - Example of User-defined Event function block

- **Event raised by a *watchdog***[8] of some variable or timer. This is the common way of reacting to variable change such as "Timer elapsed" or "Switch pressed". We have to define the watched variable and kind of the watchdog such as:
  - **Positive change**: the variable increases. It is any increase for Real variable and positive edge for Boolean. We can define an optional margin, so that the event will not fire if the change is smaller than the margin. Default margin is any change. We can also define optional time interval where the change is measured. Default interval is the previous evaluation. Default margin and interval are not displayed with the function block, but defined margin will appear at the top and default interval at the bottom of the function block.
  - **Negative change**: analogical to Positive change
  - **Any change**: Either positive or negative change, any Edge for Boolean

[8] Watchdog is a feature of a variable that monitors value of the variable and raises event if predefined change.

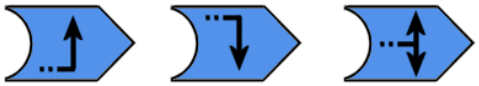


Figure 16 - Examples of analog Positive change, Negative change and Any change function blocks

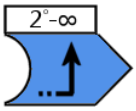


Figure 17 - Increase bigger than 2 degrees

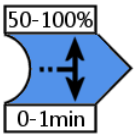


Figure 18 - Any change of bigger than 50% in less than 1 minute

- **Positive pulse**: is a Positive change followed by a Negative change. We can also define optional margins. For optional interval, we can define both minimal and maximal time. Default minimal interval is the previous evaluation and maximal interval infinity.

  For example at Figure 20, we can test whether a button has been pressed for at least 3s.

- **Negative pulse**: is a Negative change followed by a Positive change

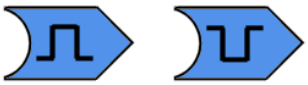


Figure 19 - Examples of analog Positive pulse, analog Negative pulse function blocks

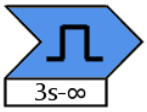


Figure 20 - Logic Positive pulse lasting at least 3 seconds

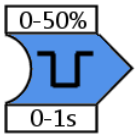


Figure 21 - Analog Negative pulse smaller than 50% in less than 1s

- **Receive email**

  Becomes true at the moment when email from one of listed senders with defined subject has been received.

Figure 22 - Example of Receive email Event

- **Receive SMS**
  Becomes true at the moment when SMS from one of listed senders with defined text has been received.

Figure 23 - Example of Receive SMS Event

There will generally be no more than one Event in one rule, because it is very unlikely that two Events happen exactly at the same time. Event can often be combined with several tests.

Events are somehow special, because they can generally have both logic and analog input. The only difference is, that for logic input we do not define the margin parameter (the margin is always logical 1) and also do not define interval for Change (the interval is always 0, meaning last iteration). We can consider, whether to have two separate function blocks for analog change and logic step, or use the same function block with left side that suggests both analog and logic input is possible (Figure 24). The Event would be displayed with this generic left side in function block library and in rule if no input is selected, but become either round or cornered depending on input type when it has been assigned.

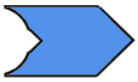


Figure 24 - Function block that can have both analog and logic input (generic left side)

Test

As Event is used to monitor change in one variable, while the Test is used for comparison of two values. The common tests are:

- **Equals:** for example "*timer = 0*" or "*Movement detected = yes*"
- **Not Equals**
- **Greater**
- **Greater or equals**
- **Smaller**
- **Smaller or equals**

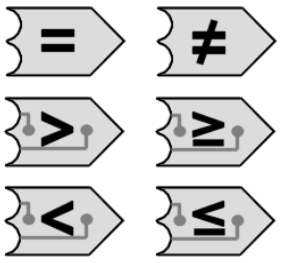


Figure 25 - Examples of Test function blocks

Naturally, we can only compare two variables of a compatible type.

We can also connect Event block (typically Positive change) after a Test block. This way we can raise event only once the Test condition has been reached.

Logic

Logic combines Boolean results of Events and test. Any logic can be achieved with a basic set of functions:

- **NOT**
- **AND**
- **OR**
- **XOR** (exclusive OR)

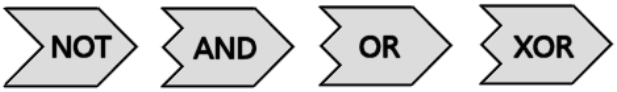


Figure 26 - Examples of Logic function blocks

Logic defined by those functions can also be translated into human language to serve as a hint what the rule does. For example: "*When Wake_up event occurs and day is weekday then immediately set garage heating to 'on' and after 30 minutes set garage heating 'off'* ", or similar to Figure 41.

Action

An Action is activated when its input is logical one. All the Action function blocks have a cornered hole on their left side, which suggests that they must be connected to logic input. Right side is flat, which means that Action is the last part of the rule, at the rightmost position. Action can:

- **Process a list of changes**. This is the most powerful and difficult part of the logic due to simplifications in other parts. The list includes items with following parameters:
  - **Name of the variable**
  - **Target value**: It can be an absolute value "20", or a relative change such as "+2" or "-1". We have to clearly distinguish negative absolute values from negative change. There is a field in between Name and Target value, which is "=" for absolute value (for example Figure 28). It also can be "-" or "+" meaning relative value change (such as increase temperature by 2 degrees at Figure 27).
    Target value can also refer to other variable.

- **Delay** (optional - only in advanced view): A relative time when the Target value should be achieved. Default value is “0” meaning immediately.
- **Transition** (optional - only in advanced view)**:** If the Interval is longer than 0, Transition specifies the curve that the variable will follow to obtain Target value from current value. Default is a step which means, that the Target Value will change suddenly at the end of the interval. Other possibilities are symbolized by an icon in between Value and Delay fields. The options include linear, exponential, logarithmic, Bezier with various parameters.

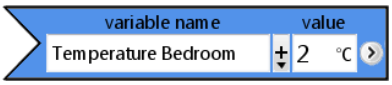


Figure 27 - Example of static list Action function block in edit mode

For easier orientation, we will display new actions in the basic view showing only Name and Target value. Interval and Transition are available in the advanced view.

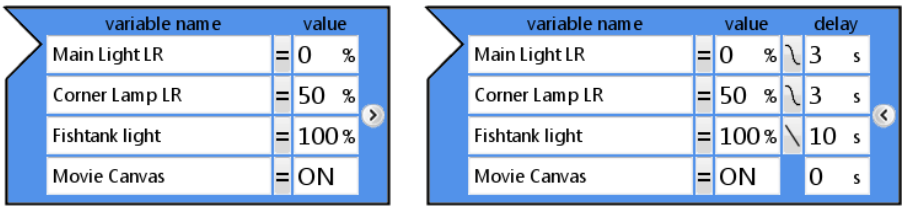


Figure 28 - Basic versus Advanced view

We can create static or dynamic lists. In a static list, the user manually adds variables to be affected each in a new line of a table. Dynamic list uses filters of type and/or area to select the devices, which are then displayed in a similar table. *We can for example leave area filter as “all” and select type “light”. All the lights will then be displayed in the table. By a checkbox we can exclude some of the devices from the list and than simply set all of the selected lights to zero. This action will turn of all the lights in a house. But what is more, whenever a new light is added to the system, user will be automatically prompted whether the light should be affected by this rule.*

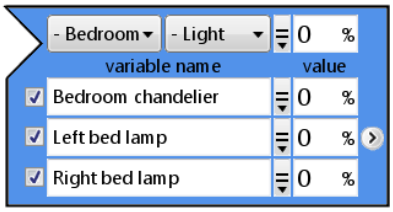


Figure 29 - Example of dynamic list Action function block in edit mode

- **Throw an Event**

Figure 30 - Throwing "Leaving the house" event, compare with Figure 15 (reacting to this Event)

- **Send email**

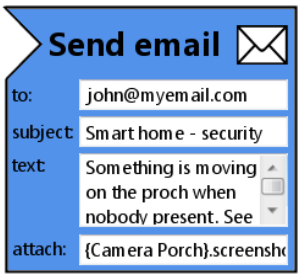


Figure 31 - Example of Send email function block

- **Send SMS** (available only if GPS module is present)

Figure 32 - Example of Send SMS function block

- **Speak** (available only if connected to audio system)
  Use text to voice engine to read out a message which can include predefined text as well as actual values of variables. We can select the areas where message is read according to the list of available audio devices.
  "**Do not read while in Sleep mode**" option is checked by default, so that the message does not disturb during sleeping.
- **Play sound:** plays selected sound / audio file

  Similar settings as Speak

- **Run script:** even though scripting is not the main control mean of middle layer in the three layer architecture, scripting should be supported for several reasons:
  - It can achieve complex functionality in middle layer, that would normally be coded within bottom layer
  - Some end-users have the skills to code scripts (or at least would like to download them), so scripting within the middle layer is safer than if they have to go to the bottom layer.
  - Some other home automation control software supports scripting, so it would be big comparative disadvantage not to have it.

  It is generally more convenient for the user to support widely used scripts than crate own syntax.
- **Command:** contains a list of other devices installed to the system and their specific commands that can be executed. Those devices has to have a driver that specifies and executes the commands.

Figure 33 - Example of Command function block

Media Center should definitely be supported by commands. We can code a simple application, that will be installed at the computer running Media Center and provide middle layer driver with access to Media Center functions such as: mute, volume up, volume down, play, pause, stop, next, previous, but also turn on[9], turn off, sleep.

A small program can also be installed on other computers within the household to support turning the computer on, off or sending to sleep.

Even though this function is very useful and fairly easy to implement, other control software neglect supporting control of other computers. In most of the current smart homes, there is a *Goodbye button* that deems to turn off all the devices and lights. It however controls devices simply by cutting the power off, which is certainly not a way for turning on/off a computer, so the user still has to switch them manually. This function is especially important for computer designated as a Media Center. Do you think that a house is really smart that it keeps playing music or a movie even when you press a button that should turn everything off?

3.3.5. Implicit versus Explicit Rules

These logic rules are sufficient to code all the basic switch logic. However, in the case of home automation, most of the logic rules are as simple as assigning a switch or timer to control a particular device. To avoid an excessive number of simple rules that would make the logic appear too complex, we will create *implicit rules*.

Explicit rules. have full definition in the logic as shown above. On the other hand, implicit rules are not explicitly defined, but they represent an ordinary link between compatible devices.

Each controllable[10] device has two optional parameters:

- **Switch:** by specifying a switch to the controllable device, we will create a simple direct link between those two. For example a switch will be directly assigned to control a light. This is also suitable for many network implementations, where we can directly assign the light to listen for signal from the switch. Both the device and its switch can still be used in explicit logic rules.

[9] Operating system provides function to shut down the computer or put it into sleep that can be accessed by a program. Another way is to use Wake on LAN (WOL) standard, which allows to turn the computer both on and off remotely. Please note that WOL depends on the motherboard and sometimes must be specifically permitted. There are many stand-alone programs, such as Poweroff that implement this functionality.

[10] Controllable device is any device that receives commands It is for example a light, a socket. Switch is not a controllable device, because it sends commands, but does not receive them and cannot be controlled.

One device ay be assigned multiple switches (*for example to implement control known as "3 way switch" or "stairway switch"* (5) - *Diagram 4*) and one switch can be assigned to multiple devices of the same kind.

- **Timer:** similarly to switch, we can assign a timer to directly turn on and off selected controllable device**.**

Implicit rules are also suitable for configuring most of the existing control networks, where direct assignment of control device to controlled device is frequently used. This topic is further discussed in chapter 0 and Figure 63.

Implicit rules are displayed as the control device(on the left side) connected to controlled device (on the right side) using a double green line. To further distinguish Implicit rules, their function blocks are squared, regardless variable type.

Figure 34 - On/off light controlled by a touch button – Implicit definition, compare with Figure 36

3.3.6. Translating Implicit rules to Explicit

It is possible to translate all Implicit rules into Explicit definition. This is also necessary for rules processing. For example Implicit rule representing connection between a dimmable light and a dimmer switch is translated using three function blocks at Figure 35. A similar scheme applies for binary switch (on/off) operating a binary device.

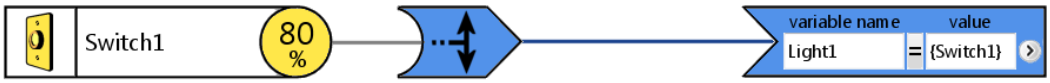


Figure 35 - Dimmable Light controlled by dimmer
(Variable Switch1 is watched by Any Change watchdog. It fires Event that assigns Light1 the same value as Switch1)

One Implicit rule may be sometimes translated into several Explicit rules. For example the value of a touch button (button, which is true as long as it is touched, but then returns to false when released) cannot be directly assigned to controlled light. Figure 36 shows how a connection between such button and a binary light can be expressed explicitly.

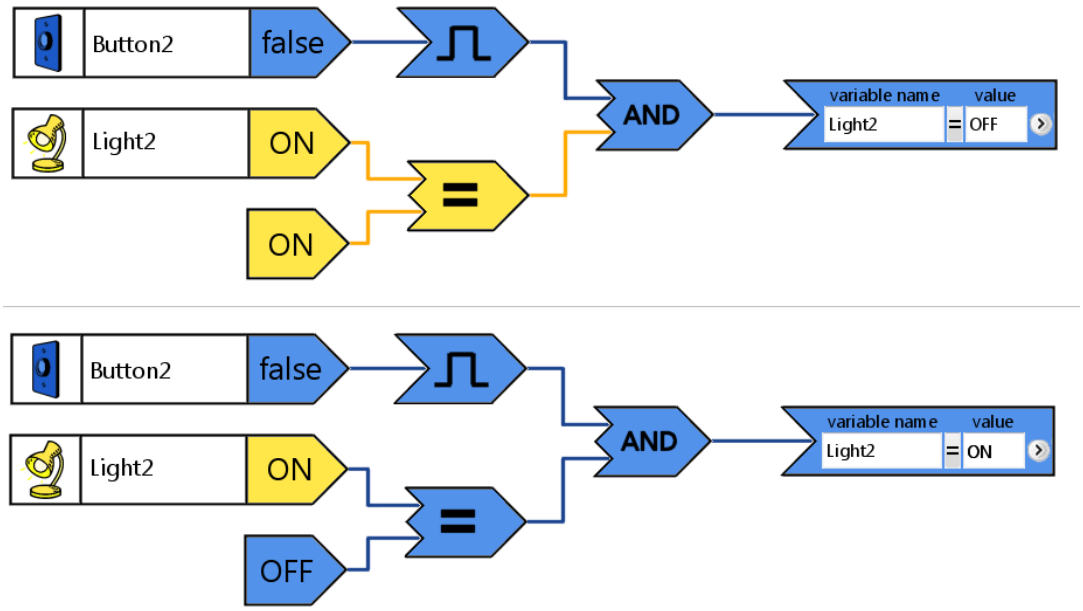


Figure 36 - On/off light controlled by a touch button – Explicit definition, compare with Figure 34 (The first rule turns the light off in case it was on, the second rule turns it on if it was off)

3.3.7. Translating rules from ordinary logic into simplified

The simplification of logic reduces functionality compared to typical bottom layer application, but some functions can be achieved by other means:

- **Propagate real variable through the rule** – Even though a real variable does not propagate through the simple logic, it is possible to achieve the same effect. We can watch for change of this variable and in resulting Action assign the same variable to any target.
- **NAND, NOR, XOR** – can be easily composed from NOT, AND and OR functions.
- **Switch on-delay / Switch off-delay** – instead of the on/off delay defined in the first part of a rule, we can use ordinary Change/Edge Event and define delay as the Interval of the following Action.

3.3.8. Processing rules order

All the rules are defined in a consecutive list. As in other systems of logic, the rules are processed in the order they are defined, which also means that latter rules can overwrite results of preceding rules. The fact that placing a rule at the end of the list can have completely different effect than placing the rule at the beginning of the list is natural to the programmer, but poses big challenge for middle layer's user and must be well explained and documented in help and manuals.

Middle layer's logic is relatively simple and most of the rules are independent, which means that their placement within the list does not matter. Nevertheless, the user may often create two rules, whose action affect the same target. A cycle appears when two rules start periodically overwriting each other's results.

For example one rule will be turning a light on, because luminance in the room is low and other rule will be turning the light off, because nobody is present.

This kind of conflict is difficult to discover from the rules itself, but we will take advantage of the affected variable's agent, which tracks history of the value for couple last cycles as we have defined in Chapter 3.2.1. If the value is periodically being overwritten (this can happen within one processing cycle or even two or more cycles depending on rule order and number of rules involved), the agent throws an alarm. In some cases this is a desired behavior and this kind of alarm will be forever silenced, but it can also reveal serious hazards in the logic that would be difficult to discover without agents.

Implicit rules are translated into explicit rules for processing the logic, but this stays transparent for the user. When the logic is executed, implicit rules are processed prior to explicit rules (Figure 37). It means that explicit rules can overwrite the result of implicit rules, which however results in a conflict solved by the agent.

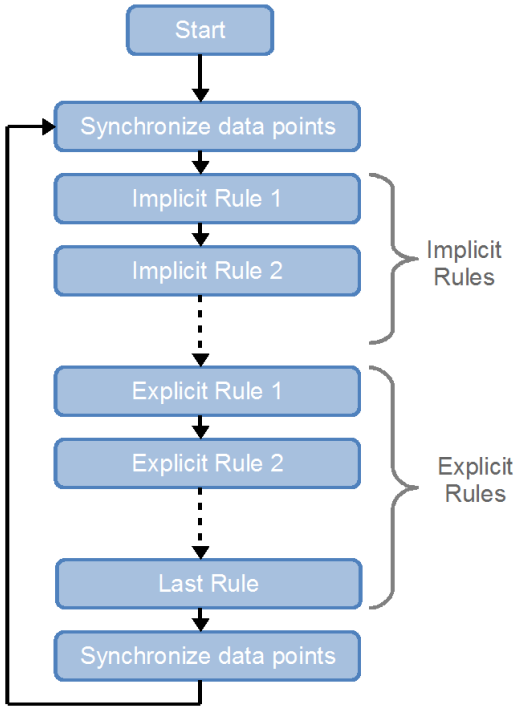


Figure 37 - Rules processing order

3.3.9. Processing a rule

Every individual rule can have many branches as we can see at Figure 38. The rule has three parts: Event/Tests, Logic and Actions. In every rule, there is a single point that separates Logic and Actions. We will call it the *Central Point* and it is also the place where algorithm for processing rules starts.

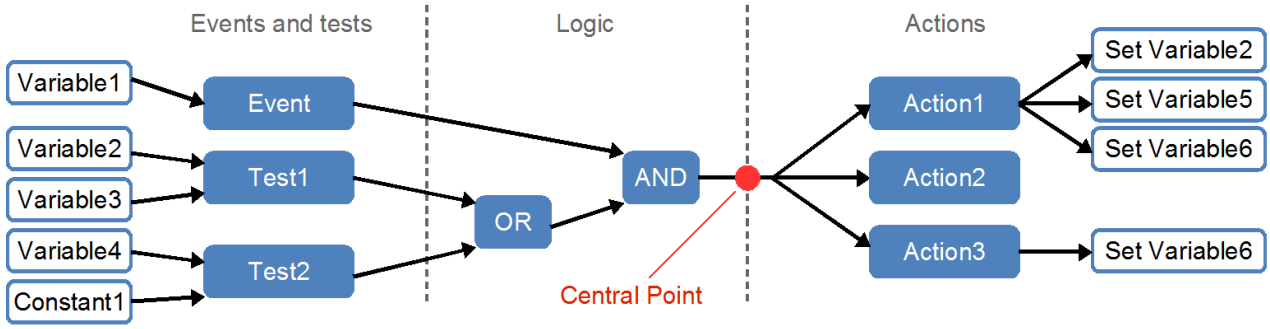


Figure 38 – Processing a rule

Algorithm for processing a rule based on backtracking:

1) Go from Central Point leftwards, expanding the logic tree and Event/Tests until the first leave of the tree is reached. This can be a Variable, Constant, Time program (which essentially also is a variable) or user defined Event.
2) Evaluate result: If Event occurs, the result is true; if the Test condition is met, the result is also true. Otherwise result is false.
3) Step back (rightwards) in the tree to the next logic element. If it is:
    a. NOT
       Change false to true and vice versa
    b. AND
       If the result is false, step back in the tree with false result.
       Otherwise evaluate next branch – go to 2) or if it was the last branch, step back with result true
    c. OR
       If result is true, step back with true result
       Otherwise evaluate next branch – go to 2) or if it was the last branch, step back with result false
4) All the logic results in Central point either into:
    a. false – the algorithm terminates
    b. true – all the actions are fired

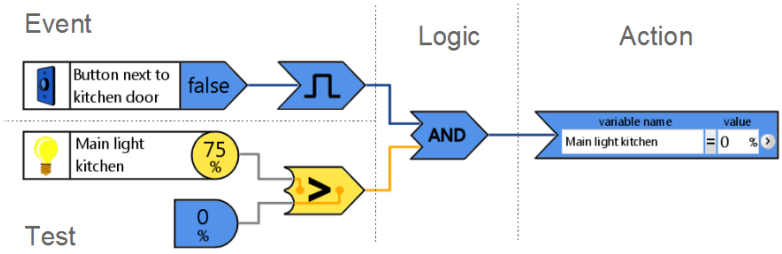
3.3.10. Data structure representing rules

Test often compares two values. Every rule typically has one Event and any number of Tests, which are then combined in Logic until the final result is reached Central Point. From this point the rule will again divide into one or more actions, having some action affect one or more variables.

At Figure 38, we may notice that the rule is composed of two trees, both with the Central Point being their root element. The *"if" tree* goes leftwards from the Central Point and contains Logic, Tests and Events. On the other side is the *"then" tree*, which represents the Actions.

We can therefore split the rule into two trees, which is a data structure suitable for storing in a XML format. It can therefore be easily stored in DOM (*DocumentObjectModel*) and transmitted over OPC XML/DA. This data representation also suits needs of the algorithm for processing rules, because it provides direct access to Central Point, the algorithm's starting point.

An example of XML representation of the rule from Figure 10:




```
<Rule id="R126" title="Turn off main light kitchen" modified_by="admin"
     timestamp="12/22/2007 13:15:02" description="">
  <IfTree>
    <Logic logic="AND">
      <Input order="1">
        <Event event="POSITIVE_PULSE">
          <Margin/>
          <Interval/>
          <Input order="1">
            <Variable id="A105"/>
          </Input>
        </Event>
      </Input>
      <Input order="2">
        <Test test="BIGGER_THAN">
          <Input order="1">
             <Variable id="A24"/>
          </Input>
          <Input order="2">
            <Constant value="0" unit="%"/>
          </Input>
        </Test>
      </Input>
    </Logic>
  </IfTree>
  <ThenTree>
    <Action action="LIST_OF_CHANGES" title="" description="">
```

```
      <Set order="1" set="EQUAL" transition="step">
        <Target>
          <Variable id="A24"/>
        </Target>
        <Value>
          <Constant value="0" unit="%"/>
        </Value>
        <Delay>
          <Constant value="0" unit="s"/>
        </Delay>
      </Set>
    </Action>
  </ThenTree>
</Rule>
```

3.3.11. Filtering and adding rules

Because the full list of rules can become overwhelmingly long, we will often filter the rules based on the area and/or type. If any variable used by the rule (either in Tests/Events or Actions) belongs to some area, then the rule belongs to this area and also to all its parent areas (in the area tree). A rule can belong to multiple areas if it contains variables from multiple areas. Thus when we filter rules by area, we will only see rules that have something to do with a particular area. Type is completely analogical.

Inserting a rule into the filtered view is a little tricky, because there might be actually many hidden rules in between the two rules in filtered view. Whether we insert new rule a) after the first one or b) before the second one looks exactly the same from the filtered view, but can result in different rule order within the full list (Figure 39) and therefore may have completely different behavior.

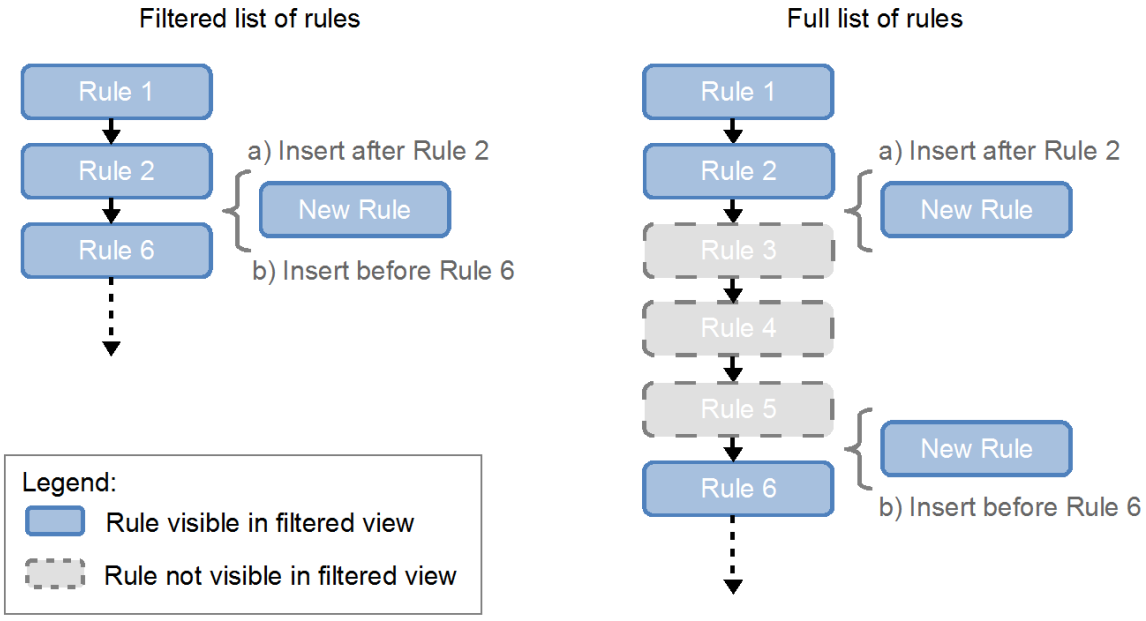


Figure 39 - Inserting a rule in filtered view

Even though this situation will occur only very seldom, we have to choose one option and the user should be aware of the choice (a pop-up message with a checkbox option "Do not show again in the future" would be appropriate). It seems that the option a) of inserting new rule directly after the first one looks more natural.

3.3.12. Downloading rules to the bottom layer

The rules are **defined** in the Middle layer and it might seem that all the rules are also **executed** in the Middle layer as described up to now. It can be the case in some installations, but some bottom layer systems and networks provide own intelligence that can also be used for executing rules to some extent. At Figure 5 (page 24) we can notice that all the green elements are connected with logic processing, but reach from Middle layer through Bottom layer all the way to the physical network.

There are good reasons to move as many rules as possible into lower level for execution:

- Rules executed in bottom layers have faster response
- System becomes more resistant to Middle layer's failure
- If all the middle layer's functionality is executed in bottom layer, we do not need the Central Computer running at all the times, but only for configuration.

Most bottom layer networks enable us to configure devices to react to signals sent over the network, so that we can directly perform many implicit, but also some explicit rules directly by configuring network devices. Sometimes, there is a smart hardware element (such as a PLC or a Data Transceiver) that connects the network to Control Computer and can execute logic operations.

Depending on capability of each particular bottom layer system, the communication driver will translate all the *independent*[11] rules the bottom layer can handle into its own logic. For determining independent rules, if actions of a rule do not affect variables used in Tests and Events of the same rule, we will virtually split this rule into consecutive rules with the same "if" tree, but only one action each. Then we can download at least the independent actions of the rule to the bottom layer.

Rules that have been transferred to Bottom layer are marked with ID of the driver that transferred the rule and are not processed in the Middle layer anymore.

Testing correct rule execution in bottom layer

Only in the case we can clearly distinguish between input and output variables (i.e. those that only occur in "if" tree versus those who only appear in "then" trees) of the independent set of rules, we can also verify if the rules are executed correctly.

We make a shadow copy of all the affected variables, but use the driver to update only input variables. Then we process those rules in middle layer on the set of shadow variables and check if real output variables, updated from the bottom layer, match their shadow counterparts. If there is not a

[11] Independent is a rule (or a set of rules), which do not share any variables with the rest of the rules and at the same time do not contain any other variables, than those mapped by the corresponding driver and its bottom layer system. Under these conditions, whether independent rule's result arrives sooner or later does not affect other rules.

match (except for a tolerable delay on either side), it means that the bottom layer does not execute rules in the same way as the logic defined in middle layer, which should result in a feedback report sent to communication driver developers and an alarm request, whether to stop executing rules in bottom layer.

It is important, that whether we download rules execution to bottom layer or not, it should be completely transparent to the user and provide the same results.

3.4. Alarm management

Good control systems have very flexible SCADA editors, which can be used to design GUIs suitable for home automation. There are, however, more differences between industrial and home usage that we have to take in account. In a factory, a supervisor is usually present all the times when the process is running. At home, there is not a person constantly watching a screen, there might be nobody present at all – and even for many days. Unlike trained supervisors in the factory, different users have different skills at home. Some alarms can be handled by everyone, some by advanced users only and others passed to an expert. This requires a completely different approach to the system of alarms.

The primary function of the alarm system is to warn about unusual and potentially dangerous behavior. The secondary function is to serve as an event log for future analysis.

In order to deal with alarms consistently and efficiently, a unified approach of handling alarms is necessary. Any part of the system can throw alarm with given properties. Alarm is an event. A series of conditions defined in the logic can result in an alarm and other rules can be defined to react on the alarm. An alarm manager is designed to catch these alarms, record them and inform user about the alarms.

The way of handling alarms in the specific bottom-layer system is essential. Some control systems have they own way of dealing with alarms, while others rely on external SCADA to take this duty. There is not a widely recognized standard for alarm exchange, so most of the time it will be up to the communication driver to facilitate alarm exchange between layers.

There is however a standard defined by OPC Foundation that the middle layer application should definitely support. *OPC Alarms and Events*(6) uses server – client architecture to share alarm data. In this scenario all the alarms are stored on a server and clients subscribe to the kind of alarms they are interested in. This way alarms caused by HVAC can be directly monitored by the system integrator, or security alarms redirected to specified SMS. All users therefore only receive alarms that are within their competence.

3.4.1. Alarm properties

In order to classify alarms, we can define the following properties:

- **Message:** the actual body of the alarm
- **State:** Active / User-Terminated / Time-Terminated / Condition-Terminated
- **Source:** source of alarm
- **Occurrence Timestamp**
- **Termination Timestamp**
- **Duration:** how long the alarm lasts - given by alarm-terminating condition
- **Category:** useful for filtering alarms (compatible with OPC Alarms and Events)
- **Area:** Location where the alarms originated, use like Event category
- **Priority**[12]**:** Composed of Importance (what is the possible threat of ignoring the alarm) and Urgency (how quickly the problem needs to be solved). This means that very urgent alarm can have higher priority than non-urgent alarm with higher importance. On the other hand, very important alarm has higher priority than an unimportant urgent alarm.

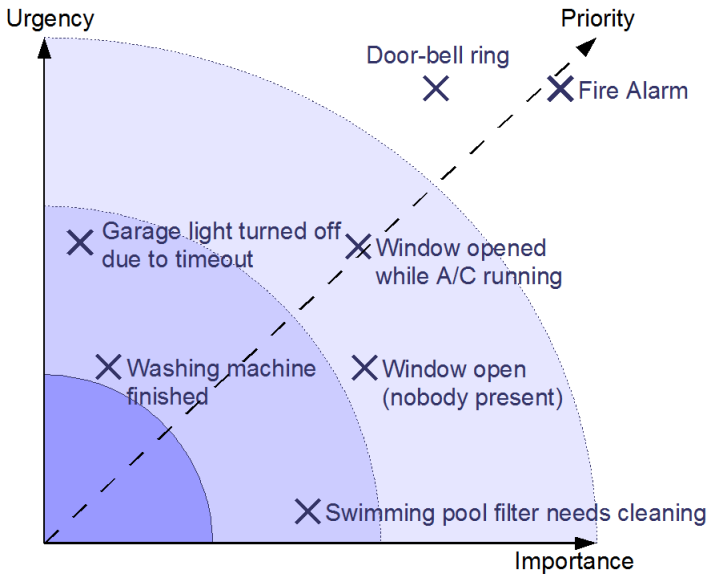


Figure 40 - Alarm priority

A good clue to setting a priority is the potential consequences if the user does not respond to the alarm. (7) We have to be careful about Informative alarms – those that do not require any corrective action. In many existing systems they only "spam" the user with distractive messages and draw attention away from important alarms. Therefore they should have the lowest priority and most of the clients can simply ignore them.

[12] Priority is also called Severity in OPC Alarms and Events. It is a number between 1 and 1000, where 1 signifies lowest priority and 1000 highest priority.

The alarm-terminating condition can be different for different alarms. There are essentially three duration types:

- **User-terminated:** alarm stays on until user confirms that it can go off. *(For example a new e-mail)*
- **Time-terminated:** alarm automatically stops after some time. *(For example a wake-up alarm)*
- **Condition-terminated:** alarm ends automatically under certain condition. Usually the same condition that caused alarm returns to normal. *(For example a power-blackout)*

Of course, user can terminate even the later two alarms. Even if they terminate automatically, they are still logged in the event history. Some automatically terminated alarms do not require any further user's attention, but others will inform the user *(For example missed call or missed door ring)*

3.4.2. Informing about alarms

Once an alarm occurs, the *alarm manager* (middle layer's client in the OPC Events and Alarms terminology) has to decide about the appropriate way to inform the user based on the alarm type and priority. As easy as it may seem on the first sight, this system can grow in complexity if required to be completely flexible. The task of the alarm manager is actually very similar to email filtering, so we can get inspired by email programs such as *gmail.com* or *Microsoft Outlook*.

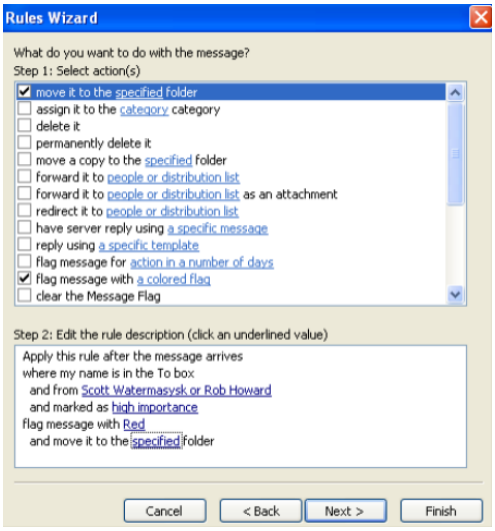


Figure 41 - Screenshot of Microsoft Outlook Rules Wizard

We will prepare a set of rules, where every rule is formed by two parts. The first part identifies particular alarm message or a group based on the alarm's priority, name, category, device that send the alarm and logic conditions. The second part is a list of actions to be performed when the alarm matches the first part. As well as in email programs, the rules can be displayed in a fairly human-like language. *Identification can be for example: "Alarm from fridge" or "Alarm with normal or higher priority and home_occupation=nobody_home".*

An action then looks like "Send email with alarm details to lily75@gmail.com and post alarm details at touch screen_in_kitchen."

The possible actions depend on the specifics of the system and will be defined by the programmer during system setup. Examples of actions include:

- **Send email to name@email.com**
- **Send SMS to +420 123456789** (if GSM module is present)
- **Send MMS to +420 123456789** (with photo from camera or recorded sound if applicable)
- **Send message to ICQ/Skype/MSN**
- **Read message at living_room** (if audio system connected to the system is present, it will read alarm message in the specified destination. The destination can be directly set to a list of spaces that have necessary audio equipment. In future, when people tracking is more common, the destination could also be determined by a person – if computer tracks movement of people around the house, it can deliver the message exactly to the room where its recipient is present)
- **Post message at entrance_hall_display** (The alarm message will be displayed at specified device)

Rules are processed in a sequential order and for each rule we can choose whether to proceed also to eventual following rules or whether to stop processing other rules if this rule has been fired.

We can use alarm_details to format the actual message of the alarm and include attachments such as recordings from security cameras.

Each alarm is assigned an alarm-terminating condition which also applies for the rule. Different actions can have different user-terminate action associated with them. For email, SMS or other messaging system it could be just a reply with “ok”, for voice announcement the user could just respond “ok” if microphone and voice recognition is installed and finally message posted on a screen can be dismissed by tapping a button.

We can also set an alarm caused by an alarm that had not been terminated within a given time. Using this scheme, we can define an easy rule to call the fire brigade when fire alarm is not terminated within one minute. This way we will avoid many unnecessary fire-alarm calls caused by smoking a cigarette or burning food in the microwave, while still keeping good protection in case of serious accident or if nobody is present to check the situation.

In order to prevent certain alarm from re-occurring in the future, an administrating user can add this alarm on a black list. He can choose whether to ban the alarm completely or only temporarily, for a given time. This is an efficient way of reducing alarm overload and is frequently used by anti-virus software. We can also let the user to add a comment to the banned alarm in order to get oriented in the black list later on. Because the system of logic throws an alarm whenever a variable is being systematically overwritten by more rules within one cycle, we can easily disable this warning for cases when this behavior is correct.

The advantage of OPC server / client configuration is that we can allow third-party clients to handle special alarms that are not possible to handle by simple middle layer’s alarm manager.

There is a number of theories and applications for alarm handling, which go beyond the range of this thesis (8) (9). They cover topics as concurrent exceptions, fault-tolerance and flowchart diagrams.

3.5. User interface

If the end user opens just about any bottom layer logic software, he will see an abundance of panels, toolbars and controls that he does not really understand. Those are handy for an experienced programmer, but would rather scare the normal user. Current specialized home automation software usually show easier interface than general control software, but they are mostly text-based and difficult to get oriented.

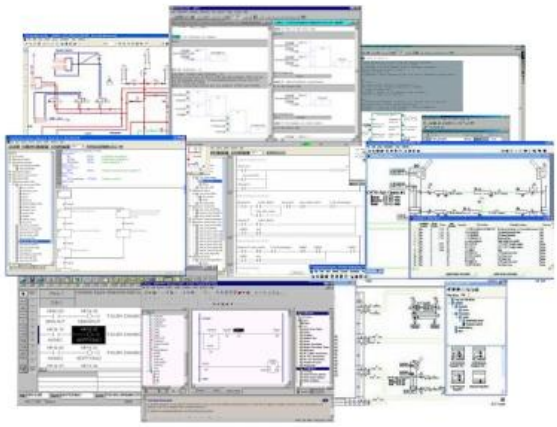


Figure 42 - Complicated bottom layer system

The first step would be to limit the number of controls to the minimum and organize them in an intuitive manner. Because we do not want to confuse the end-user with advanced controls and settings, but we need to enable to access those features, we can use the advantage of a *dual view*. Many dialogs and controls can have two kinds of user interface:

- **Basic view:** maintains simple look with only the basic options available. There is an option (i.e. expanding arrow) to enable the Advanced view.
- **Advanced view:** features all the functionality and available control. There is an option to switch back to Basic view.

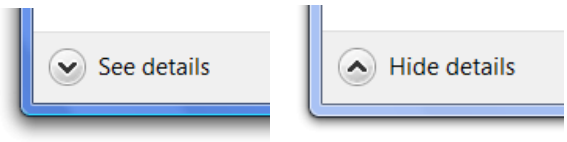


Figure 43 - Example of Basic and Advanced view in Windows Vista

The user can choose what view to open by default:

- Always open Basic view
- Always open Advanced view
- Open the view used last time

Icons

The icons and symbols used by experienced programmers may not be understood by end user, so we have to modify the graphic design as well.

Help

Obviously, we need to provide good help support and on hover hints. The same help content an experienced programmer would appreciate is not suitable for an end-user. Help explanations must explain the subject clearly without relying on user's control system knowledge. There can be a typical example included for most of the help entries.

Undo / Redo / Repeat

An immediate Undo / Redo is already a necessary part of a comfortable interface. We can also consider the Repeat[13] option, which is becoming increasingly popular. In most applications, Undo is only available for current session, but Undo information is lost when the application is closed and opened again. Also, normally if we undo some steps and then make changes, we cannot get back to the original state before Undo. In middle layer we will go even further and track and archive all the changes, so that an expert can help the user to recover from mistakes even later on.

Remote GUI

According to current programming trends, GUI is separated from the code itself. We will design a remote GUI so that we can connect to the local control system from anywhere. This concept has several advantages:

- The control computer itself does not have to have its own graphic output; we can connect to it from any PC/laptop.
- Technical support can conveniently access the control system without actually having to come to the house. This way a grandma can call her grand-son to adjust the lawn sprinklers simply from his dorm. Also, the cost for system provider's "official" technical support can be lower.
- More people can connect to the system at the same time, for example while discussing the situation over a phone.
- We can design platform dependent GUIs. We do not have to be limited to GUI of the platform where code is executed.

We can deal with the disadvantages:

- Communication is slower than if the GUI is together with application.
  In smart home middle layer, the speed is actually not essential. It is okay if we have 200ms delay after somebody turns on a switch before we can see it.
- Hazard of more people doing changes at one time. We can easily limit this by allowing only one user to have the write permission at one time, the others can watch in "read-only" mode.

Currently popular way of achieving this remote GUI is a web-server. HTML guarantees that everybody can simply connect to the server. As accessible as it is, HTML has its limits and can hardly provide as good remote interface as a local one could be. It is a good back-up option, but we might want to look for some more advanced technology to serve as the main interface. We can use for example the new *XAML*

[13] Repeat function can often be found instead of Redo at times when there is nothing to redo. It repeats the last action performed on the system – for example insert the same function block that has just been inserted.

(Extensible Application Markup Language) for programming .NET applications under the *WPF* (Windows Presentation Foundation), which bridges web and desktop applications[14]. XAML with its attractive graphic features and separation from the actual code is an ideal interface programming language for future smart home control systems and it is also used in this thesis to demonstrate the GUI.

Note: Later in this chapter a series of screenshots of a middle-layer application GUI is presented. The sample application was created in Visual Studio 2005 using WPF and XAML technologies. At current state, it is not a fully functional application, but rather a visual demonstration of ideas presented in this thesis. A commercially successful middle layer application would need professional styling and graphics, so please do not consider this sample application to resemble the final product, but merely as a source of inspiration.

The GUI consists of four basic views:

- **Home**– displays basic information about current state, alarm messages, update information and other custom information such as clock, calendar, latest news or weather forecast. It serves as a hub for other controls and can be much customized to serve particular client's needs.
- **Table** – offers a table view of all the connections and their data points. It enables data point management – adding, editing, mapping...
- **Scheme**– is a graphical ground-plan or a picture of the home. Data points are positioned on the scheme and shown as gadgets that represent their state. *For example, a light can be represented by a bulb image that is lit up if the corresponding light is on.* It is used to visualize state of the building, organize data-points by area and find data points in an intuitive way.
- **Logic** – displays logical rules. Those can be filtered by type and area.

We can display only one of those views in the full-screen, or multiple in the spit-screen mode. This feature will let us see corresponding data between the views and use drag-and-drop.

*For example, we can easily locate the device that caused alarm displayed at the Home view in the Scheme (*Figure 45*). Or we can view all the data points associated with some rule in Logic highlighted in Scheme or Table. We can also build the Scheme easily by dragging data points from the table and so on…*

[14]For a proof that the power of desktop applications is not limited to desktop anymore, but can be accessed via internet browser easily, please see the Windows Vista emulator project at http://www.vista.si/main.htm

This application has been created using XAML and SilverLight, which is Microsoft's new response to Flash.

3.5.1. Home

The Home View page appears when the application is started and it is fully customizable using gadgets in a fashion very similar to My Google homepage (Figure 44). There are default gadgets displaying important information, but new third-party gadgets can be added to support control of additional devices attached to the system or bring additional features. User can freely move gadgets around the page, change their size and attributes or minimize those, that he does not want to see.

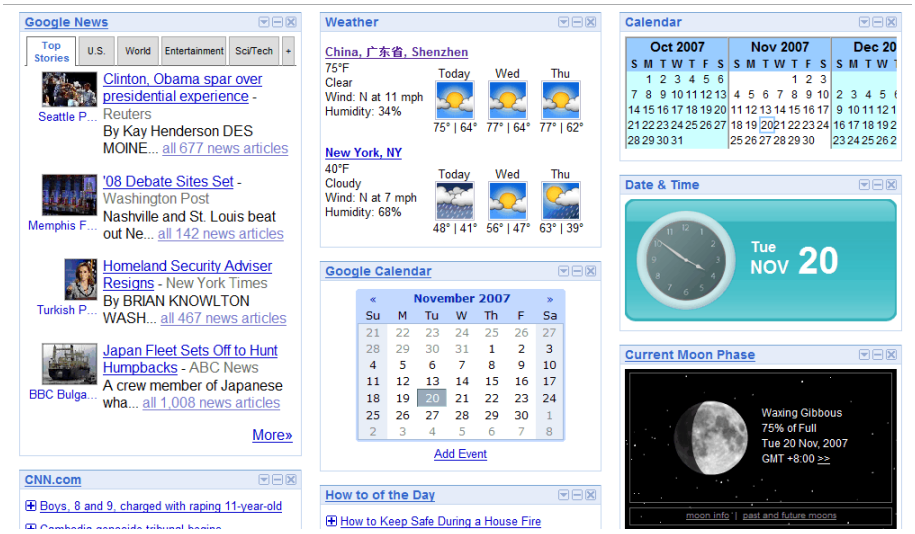


Figure 44 - My Google homepage (www.google.com/ig)- similiar to Home View gadgets

Default gadgets:

- **Alarm log** – view of recent alarms. We can click an alarm and directly see corresponding rules, data points or appliances highlighted in the spit screen view. We can view details, confirm alarms or forward them to responsible people. More on alarm management in chapter 3.4.
- **Event log** – view of recent events. Similarly to Alarms, we can click events to view details and directly find their source.
- **Upcoming Events** – similar to Event log, but displays planned events based on time schedules. *For example in Upcoming Events we can see that in 10 minutes the sprinklers will run and in 2 hours decoration lights turn on. After 10 minutes, running the sprinklers will move to Event Log, but in the meantime there may already be some new events, such as pressed buttons or unlocked doors – events that are not based on schedule.*

We can display all Alarms and Events in a table or pinned on a timeline graph.

- **Action shortcuts** – we may place buttons for executing some action, such as changing a light scene, directly on the home page.
- **Security webcam picture** – gadget that directly displays a security webcam picture.
- **Power consumption** – shows current power consumption and plot historical power consumption data. See 5.1 - Power consumption optimization.

We can also add gadgets for monitoring and controlling other devices connected to the system. We can have a gadget for sprinklers that will show watering plan according to weather forecast, gadget displaying news about home automation products and so on…

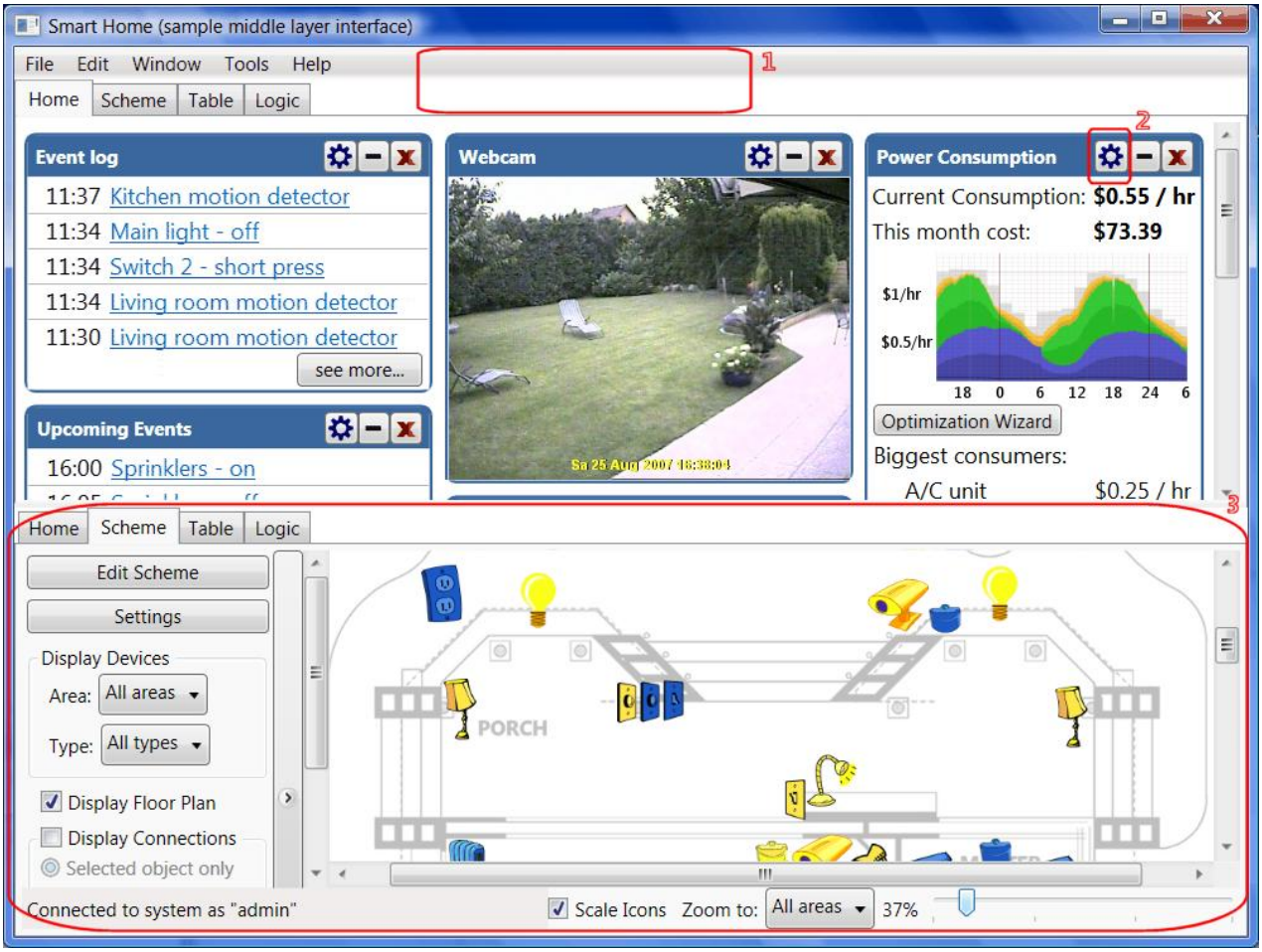


Figure 45 - Home view with split screen Scheme view

1) **In this area of the screen could be placed contextual menu such as in Microsoft Office 2007 – applies for all views**
2) **Button for setting gadget properties**
3) **Split screen allows to locate devices mentioned in home view**

3.5.2. Scheme

Scheme is a graphical view of the control system. The aim is to provide similar functionality that Table view offers in a more intuitive visual fashion and also make locating Variables easier. A number of users can be connected to the scheme for viewing but only 1 user a time can be in the edit mode to make changes.

Bitmap image or images (typically floor plan of the building or digital photographs of actual place) are placed on infinite canvas to form a background, which is usually set up by the system integrator. He then selects polygons representing Areas, which can be used for zooming to particular Area or for gadget placement. Labels (such as Room names) can be placed on the picture for easier orientation.

Gadgets represent Variables by an icon, which can be assigned from a library. Icons should be simple and using tones of one basic color, so that the color scheme of the whole gadget can be changed depending on the state. Blue color in this example represents inactive state, while yellow color stands for active state of the Variable. Green color has been used for devices connected with control network and red for devices with some error. In the sample application, bitmaps have been used for icons, but it would be advisable to use XAML vector graphic for smooth scaling and switching the colors.

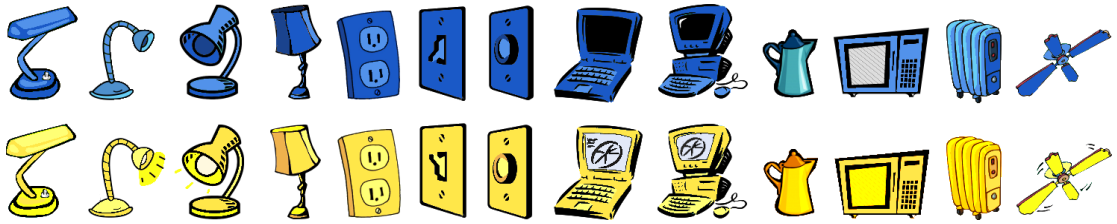


Figure 46 - Sample gadget icons

Gadgets are positioned on the plan either from a pop-up menu in the scheme or using drag&drop from Table view.

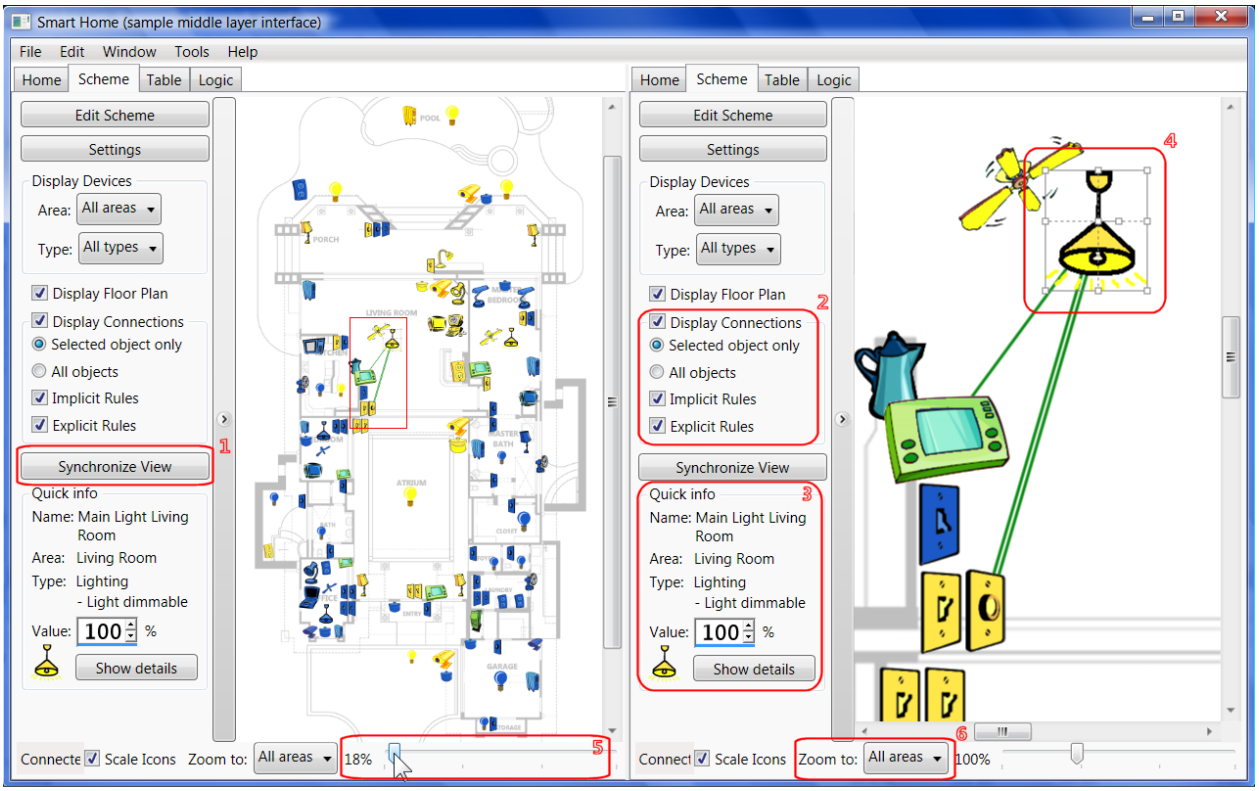


Figure 47 - Scheme view

1) Synchronize View button is only available in split screen mode and it synchronizes the view settings and zoom between the two screens. In split view mode, the active area of the other screen is highlighted in a red rectangle.
2) We can display connection of selected or all objects given by the rules. There is a connection between gadgets if both Variables represented by gadget are present in one rule in the logic. Explicit rules are shown by a single green line and Implicit rules by double green line. In the sample application green color is used to display control system and network elements including logic connections. We can directly view the rule definition in logic in split screen view.
3) Quick information and control of the selected gadget is available directly on the panel.
4) Gadget can be freely resized or rotated. Dotted lines specify it's central point, which is used for rotating, scaling, and displaying connections.
5) We can zoom the whole canvas to get the big picture or view details.
6) We can also zoom and center canvas to get the best view of selected Area.

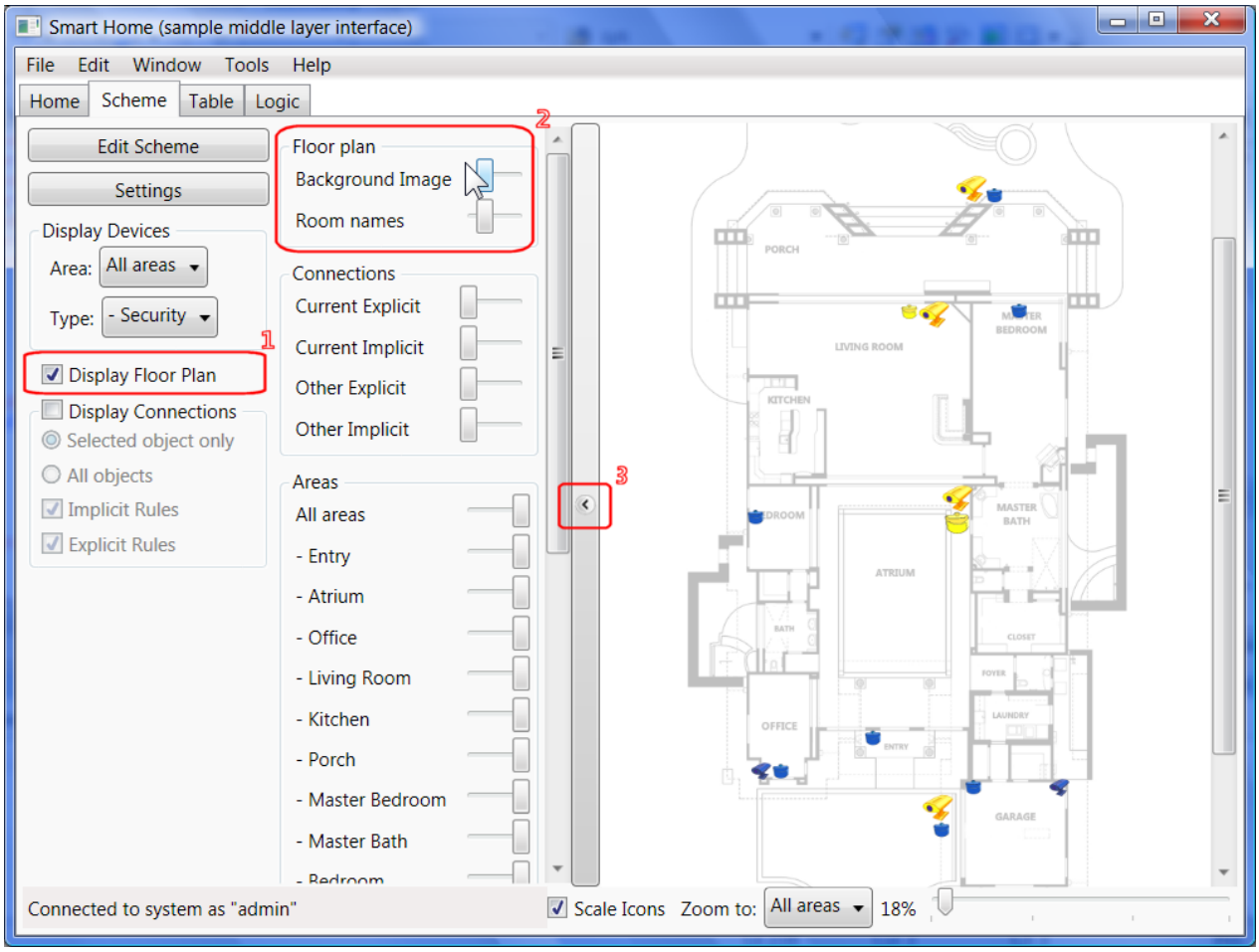


Figure 48 – Scheme, basic and advanced view

1) In basic view, we only have simple settings about elements being displayed in the scheme
2) In advanced view, we can set exact aplha (transparency) property of all element types in the scheme. In this example we fade black-and-white floor plan not to disturb other elements.
3) More/Less button allows us to switch between basic and advanced view.

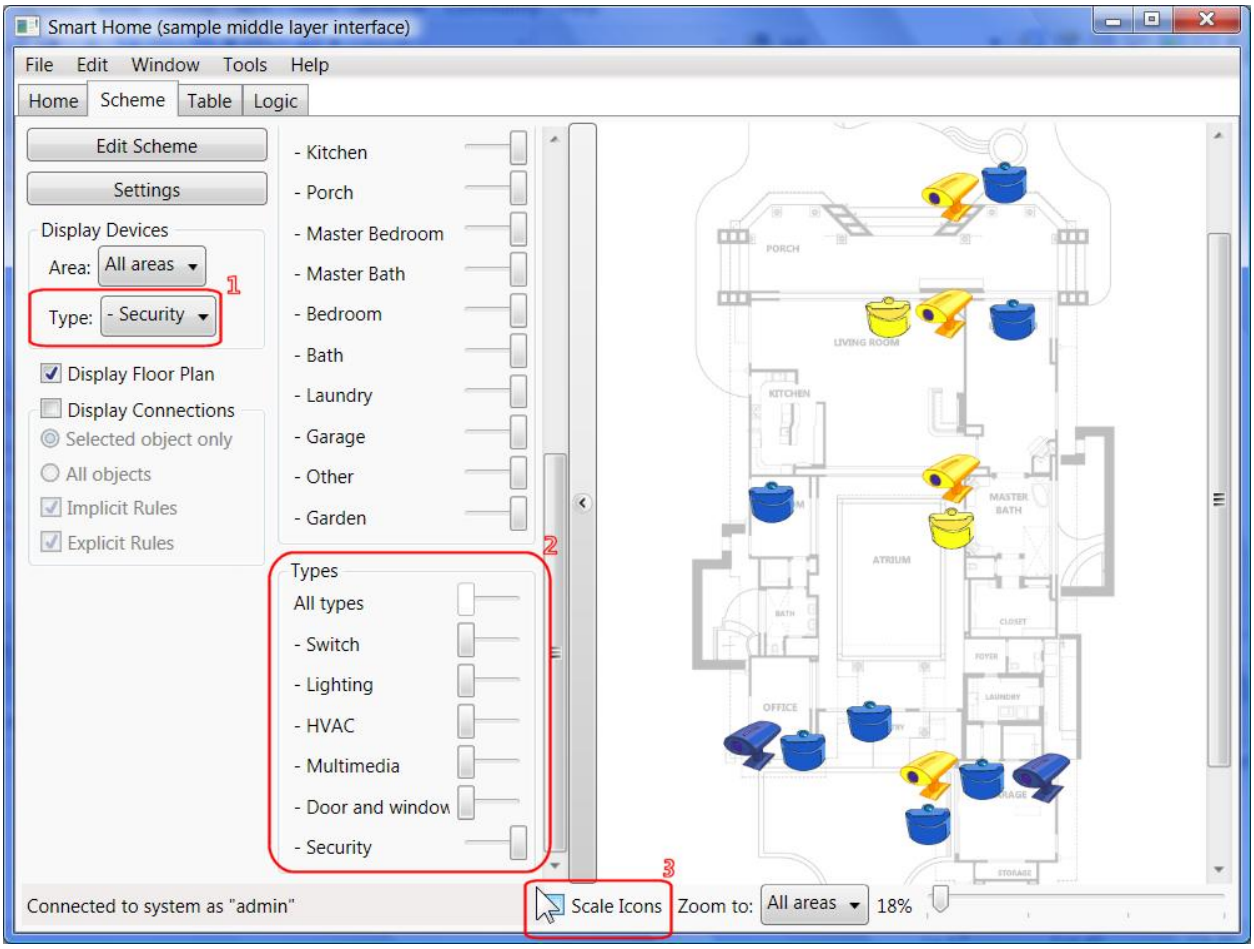


Figure 49 - Scheme view filtering and scaling icons

1) **In basic view, we can filter out all but one Type (for example Security devices) for easier orientation**
2) **In advanced view, we can choose to alpha property for any number of user-defined Types. Parent node of the directory tree controls alpha of its children. In this case "All types" is the only parent and the slider is faded, because not all of it's children have the same intensity.**
3) **We can choose the gadget icons to maintain constant size (like here) or scale with the floor plan (like Figure 48)**

3.5.3. Table

Table view provides well-organized management of Variables. In edit mode, it is possible to add, edit or delete Variables. All the Variables are displayed in a table with customizable columns – we can change their width, order and hide those that are not used, or sort Variables according to one column.

The columns include the Variable's:

- **Area** - table can be filtered according to Area
- **Type** - table can be filtered according to Type
- **Name** - only rules that match some Name search criteria can be filtered
- **Mapping** - ID of connection and connection quality
- **Bottom layer ID** - Variable's ID/name in the bottom layer
- **Last update** - timestamp of the last time when variable has been updated
- **Type** - Data type of the Variable
- **Value** - displays value with background color signifying high or low state if applicable
- **Unit**
- **Control** - allows direct control of Variables, dependent on the Type
- **History -** small graph that shows recent history of the Variable, can be clicked to access full history from the database
- **Devices** - shows icons of devices that represent the Variable in the Scheme. Device icon can be clicked to show respective device in the Scheme in split screen mode.
- **Implicit Rules** - icon for each Implicit rule, in which the Variable is included. Can be clicked to view the rule either in Logic view or highlighted in Scheme view in split screen mode.
- **Explicit Rules** - similar to Implicit Rules

View Details button will call a dialog with detailed information about the variable organized in type-dependent tabs, including:

- **Basic**
- **Mapping**
- **History**
- **Devices** - Scheme devices that refer to this variable
- **Rules**

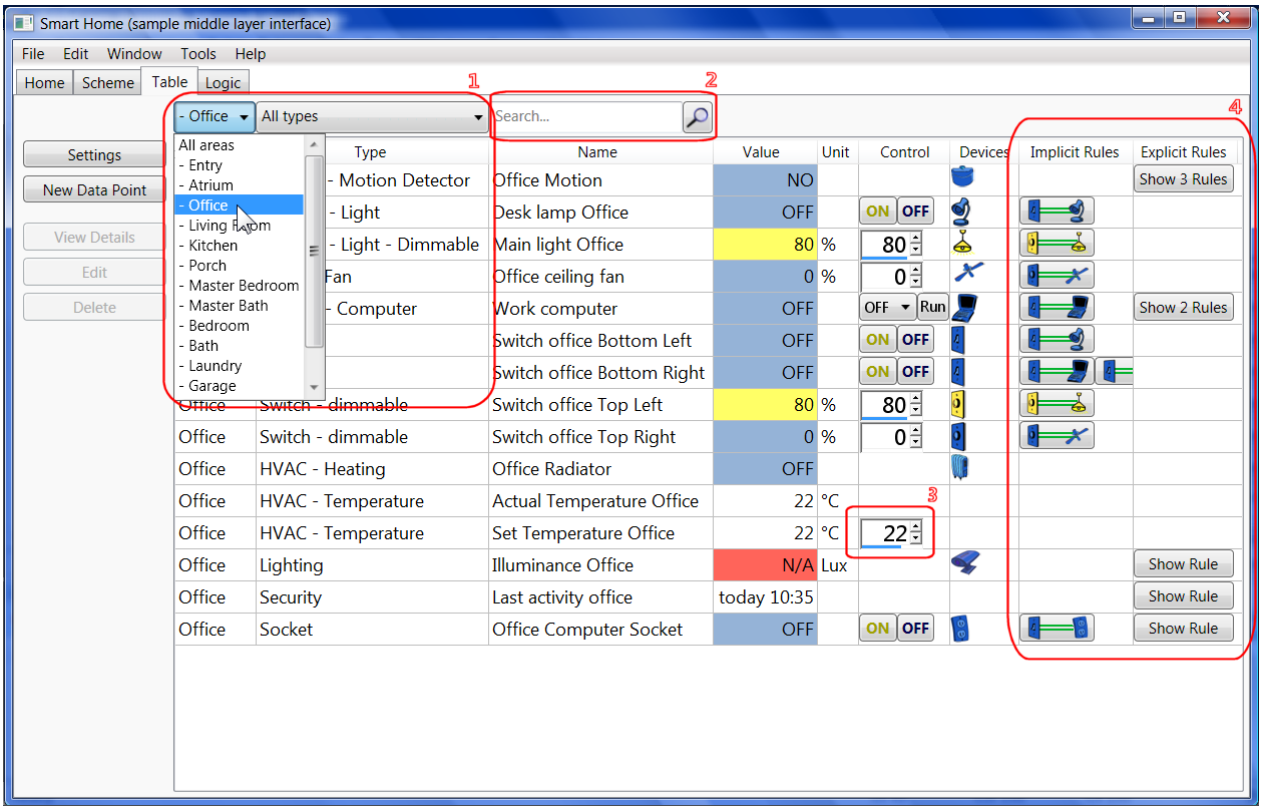


Figure 50 - Table view in Edit mode

1) **Filtering according to Area and Type**
2) **Filter rules that include searched text in their Name**
3) **Analog values can be entered directly by keyboard, increased or decreased by up and down button or set by dragging the scroller line below the number**
4) **Related rules can be directly focused in Scheme or Logic view**

Split view enables the user to drag-and-drop Variables from the Table directly into Scheme or view detailed information about a group of Scheme devices in the Table.

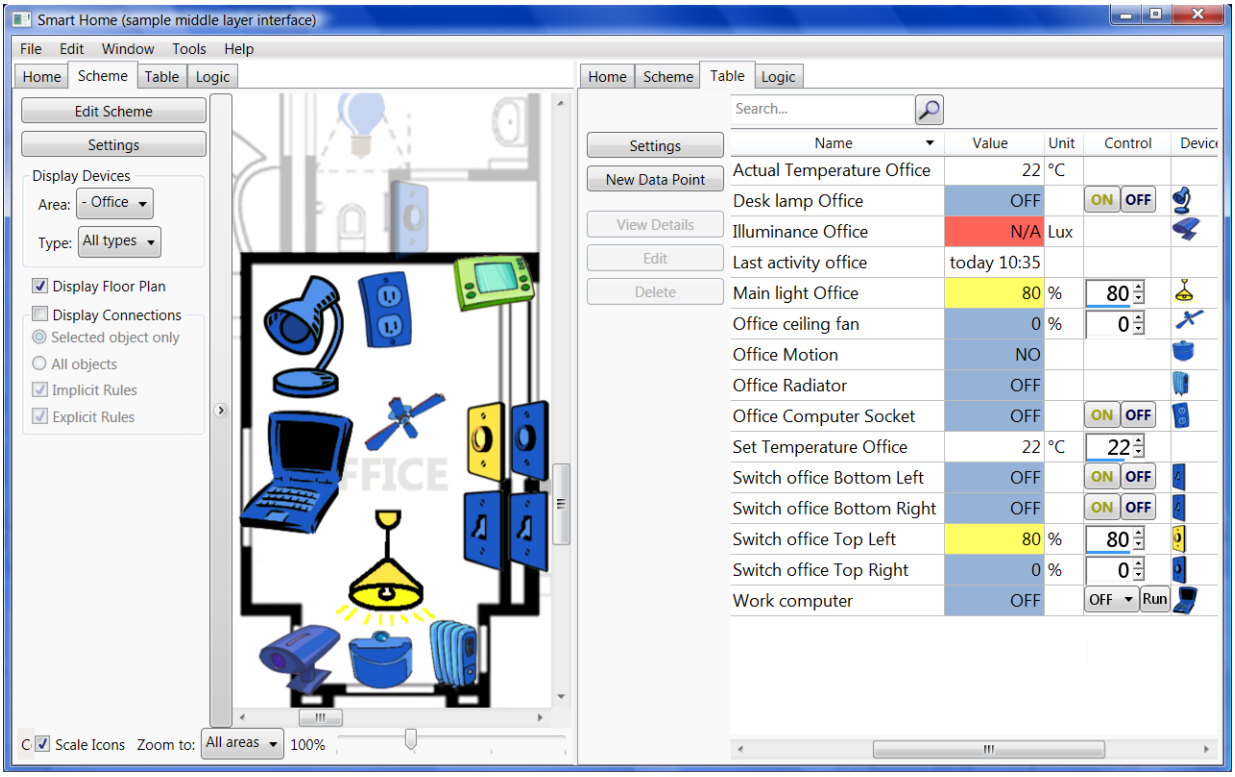


Figure 51 - Table view with split screen Scheme view

3.5.4. Logic

In Logic view we can create, view and edit logic rules. Rules are composed from function blocks that represent Variables (including Data points, Time programs), Events, Tests, Logic operations and Actions. Those blocks are connected by links that represent logic connections. Every rule consists of three parts and each part can contain only specific function blocks:

- **Tests and Events**: Variable followed by Event, One or more Variables followed by Test, or user defined Event. Instead of Variable can also be Analog function.
- **Logic**: Logic operands
- **Action**: Action types defined in Chapter 3.3.4 at page 35.

View mode

The first time we open Logic view, we can see the list of rules and actual state of their components. Tests and links that are currently “true” are highlighted, and events and actions blink highlighted for a short time when they are executed. We can filter rules according to area and type in the same manner as in other views.

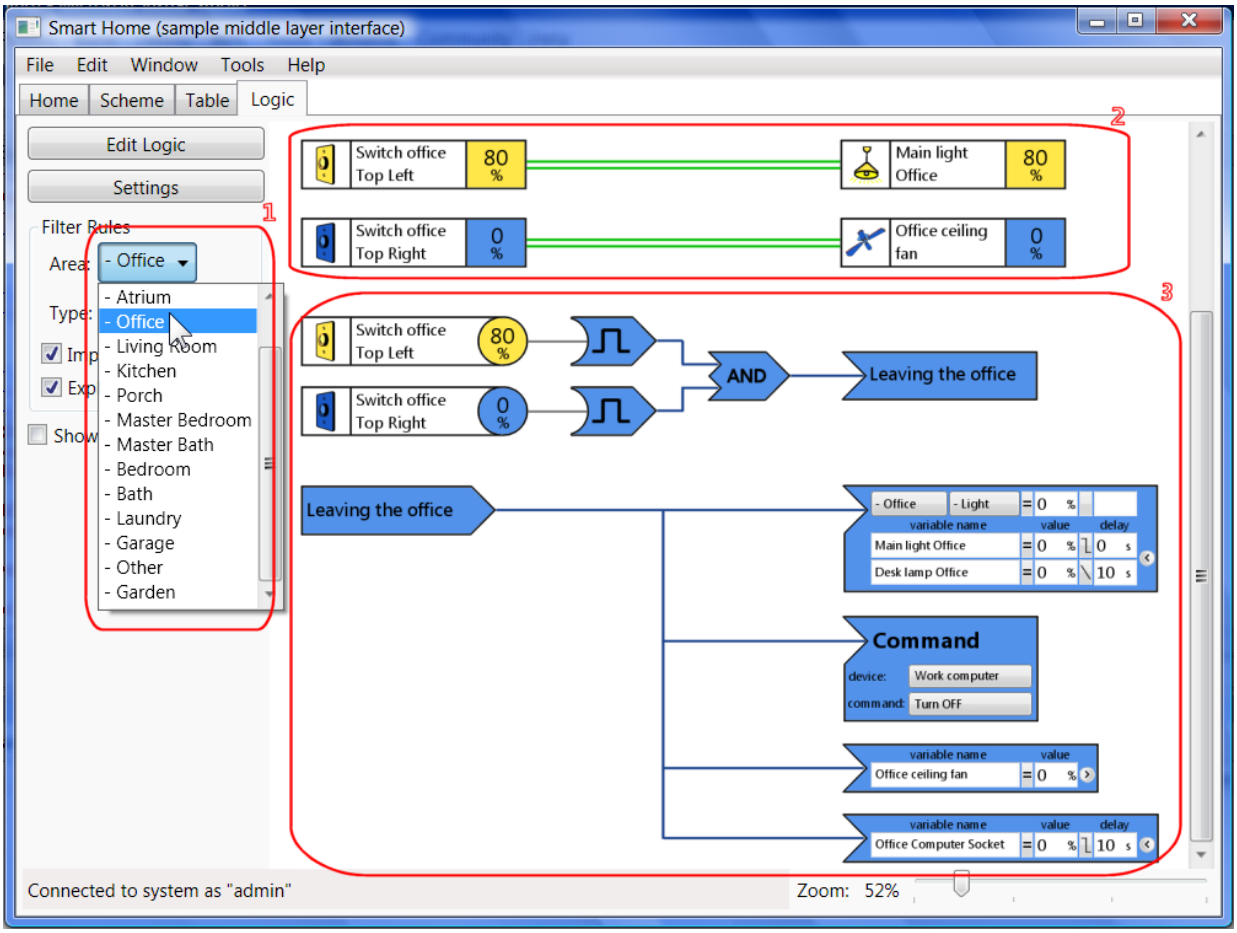


Figure 52 - Logic view filtering

1) Only rules containing variables of some Type or present in specific Area are shown, others are filtered out
2) Implicit rules are displayed first
3) Explicit rules

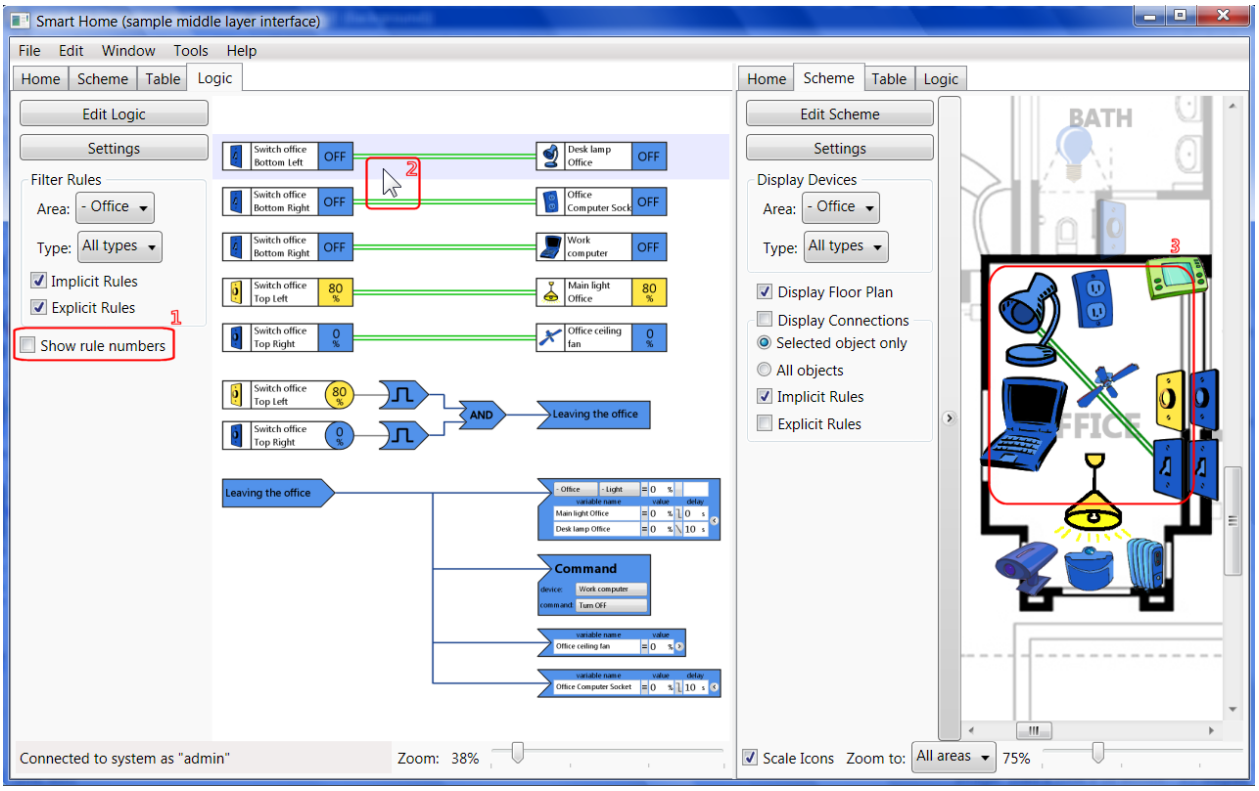


Figure 53 – Logic view with split screen Scheme view

1) We can choose to Show rule numbers to see where the rules belong in the full non-filtered list
2) We can hihlight rule in Scheme view by selecting it in Logic view
3) We could also select rule in Scheme to focus it in Logic view.

Edit mode

Similarly to Scheme view, only one user at a time can enter the edit mode. In this mode, he can add, edit or delete the rules. It is still possible to filter the rules and view their current state. There are two ways of working with rules:

- **Compose from library** – a standard way of creating and editing rules in other control systems. We have a library of function blocks sorted by categories in a tree (Variables, Events, Tests, Logic, Actions). We can drag their icons, freely place them on the canvas and create links between them by clicking and dragging the mouse. To make searching for function blocks faster, we can place icons for frequently (most recently) used function blocks on a toolbar.
- **Rule wizard** – a simple clickable wizard that guides the user through the process of creating a rule. It instantly generates the respective rule in a graphical view composed of function blocks and also generates textual equivalent of the rule in side windows. This provides great feedback for the user to check whether he has created what he intended and also teaches him to understand the block representation of the rule. The rule can be created in four simple steps (some of which can repeat a couple times):
    1. **Choose Variable** – by selecting from a list, table view or scheme view.
    2. **Assign Test or Event** – if Test involves another Variable, then choose it (step 1)
    3. **Combine results in logic** – select if the rule is complete (continue to step 4), or negate the result (insert NOT and repeat step 3), or add another condition using AND, OR, XOR (repeat step 1 and 2)
    4. **Define Action** – may also involve choosing Variables (step 1)

3.6. Connection with bottom layer

Middle layer does not have any direct connection with the physical system. All the communication goes through bottom layer application. This architecture allows for abstract constructions and relative independence on technology used in physical layer.

There are several possibilities for communication between the layers and different approaches may be necessary for different bottom layer application.

- **OPC server/client**: is a safe and convenient way supported by many applications.
- **Direct memory sharing**: if both layers run on the same machine, they can directly pass data objects in the memory. This option is not as safe, but works very efficiently if coded properly.

3.6.1. Communication driver

Because the actual data point properties and representations vary amongst different bottom layer applications, we need an interface which translates the representation we will use in middle layer to other systems and vice versa. We can call this interface a *communication driver*. Thus, defining multiple drivers allows us to connect to multiple bottom layer applications. We can also connect to more bottom layer applications at one time, which can prove handy in installations, where several bottom layer applications run in parallel to control different parts of the system.

Every layer can run at a different system, be developed in different programming environment and use different data interpretation. Thus, a robust interface is necessary to ensure smooth communication without errors resulting from data misinterpretation. In order to make the rest of the middle layer independent on particular implementation of other layers, we need to develop a special communication driver for each supported application. We should leave the architecture open, so that anybody can add drivers to support new applications.

The driver is responsible for initiating and maintaining connection and properly reflecting changes on both sides. Simplified operation of the driver can be written in a thread:

1) Initialize connection
2) While (not end)
    i. If (variable changed in other layer) then update variable in middle layer
    ii. Run middle layer logic – synchronized with other drivers
    iii. If (variable changed in middle layer) then update variable in other layer
    iv. Synchronize alarms
    v. If (rules are modified) download rules from middle to bottom layer
3) Terminate connection

The driver has to follow the priorities as set in 3.7 and raise alarm in case of violation as discussed in 3.4.

We need to communicate three kinds of data:

1. Variables (Data points)
2. Alarms
3. Logic Rules

3.6.2. Data mapping

Data mapping is the procedure of passing variables between layers. It is relatively easy, because we only need to define mapping rules, which correctly translate a table of variables. The key points are to provide a superset of commonly used data-types in the middle layer and use high precision data-types to minimize the risk of overflows and other errors. OPC XML/DA is a vendor-neutral Web service that can be used as the main standard, because major home automation systems are expected to support this standard.

3.6.3. Sharing alarms

There is not a widely used standard for sharing alarm information, but it is reasonable to support at least *OPC Alarms and Events*(6) in the middle layer.

3.6.4. Downloading logic

The more logic rules defined in the middle layer we can download for execution in the bottom layer, the better, because then middle layer application is only required when making changes, but not during normal operation. Some distributed bottom layer networks provide each device with own intelligence and some systems rely on a PLC to host control algorithms.

The most convenient situation would be if all the logic defined in middle layer could be directly translated into the bottom layer hardware such as a control PLC or intelligence of a distributed network, so that middle layer application could be used only for network configuration, but not required online. This approach is today widely used for basic home automation, but advanced functions that require more processing power (*such as optimizations, audio/video processing and functions with information available from the Internet*) essentially need some more powerful machine. It results in a PC based control computer as described in Chapter 4-Hardware structure, where the middle layer runs at all the times.

Simple logic rules can be usually translated to almost any widely used bottom layer system. Because middle layer is independent on the actual bottom layer, but each bottom layer system has different capabilities, it is up to the communication driver which rules it is able to download into the bottom layer and which functions will remain in the middle layer. If we have a machine, where middle layer application runs all the time, this translation is transparent to the user. A suitable hardware structure will be further discussed in Chapter 4.

There is even less standardization for logic rules than for alarms or variables, thus each driver has to implement bottom layer – specific translation. Since the data structure designed for storing and interpreting the rules in Chapter 3.3.10 uses XML, we can also share rules using OPC XML/DA.

3.6.5. Conclusion

We can use standards set by OPC Foundation to share both Variables, Alarms and Rules, but the particular way of transferring data depends on each bottom layer system and is defined in communication drivers.

3.7. Priority

What happens if two layers want to control the same variable in a different way? What if the user acts against the rules defined in logic? In this chapter we will define priority to answer those questions.

3.7.1. Layer – Control Field separation

All the vital control algorithms are coded in the bottom layer and therefore it has to have higher priority than the middle layer. We can restrict access to some data points in bottom layer. To avoid conflicts, data points that are exposed as read/write for middle layer had better not be modified from the bottom layer. Data points which can be changed from the bottom layer should appear as read-only for the middle-layer. This is, however, dependant on particular bottom layer system. Middle layer has to maintain stable functionality and react to read/write variables changed from the bottom layer. It can collect a list of those variables and display as a warning. *For example, if there is a rule to turn off garden lights during the day in the bottom layer, but user for some reason wants to turn them on, the bottom layer's rule has higher priority and therefore will turn them off, which could confuse the middle layer's logic.*

In order to avoid priority conflicts, we should define a clear separation of consistent Control Fields within layers. This means that all the rules connected with some area of control should be implemented in the same layer. In case of home automation, **we will code all the HVAC algorithms in the bottom layer**. Correct synthesis of all the rules connected with heating system requires expert knowledge that a common user does not have. The end user will only be provided with simple interface such as the target temperature in every room and settings whether he prefers rather economic heating or more comfort. On the other hand, we can **leave all the logic connected with lighting up to the user in the middle layer**.

Example: *In bottom layer application, we have a reasonable rule to turn the air-conditioning off when window is opened. In middle layer, the user creates a rule that the air-conditioning should be turned on if the air temperature exceeds 25°C. What happens if there is 26°C and the window had been open?*

Nothing. Middle layer will try to turn on the air-conditioning, but bottom layer will not allow this since the window is open. This will result in a warning displayed from the middle layer and user's dissatisfaction. What went wrong?

In this scenario, we are controlling one subsystem both from bottom and middle layer. We did not follow the Layer – Control Field separation. To fix this problem, all the rules for heating should be defined in the bottom layer. User will just set maximum temperature to 25°C, but he wouldn't have to care how to obtain this state – it is up to the intelligent bottom-layer algorithms.

3.7.2. User's priority

In home automation, there are hundreds of opportunities to use artificial intelligence – it can often be made by simple logical rules.

- If window is opened, turn the air-conditioning off.
- If the user is present, and lighting in the office is not sufficient, turn the lights on.

Seemingly simple and robust rules can make the living space feel more intelligent. Even the smartest system integrator might not be able to estimate all the connections, impact and undesired situations, which can result from combinations of seemingly harmless rules and end-users actions. You know the feeling, when a super smart machine with only one button actually becomes stupid if you want it to do a little different task that its engineers did not think of.

It is, therefore, good to implement this kind of intelligence in the middle layer, so that the user can view, change or disable it by himself. **Since bottom layer has the highest priority, no questionable rules should be implemented there**.

There is also a risk that a set of simple and innocent rules can get into deadlock or other unstable state. What if a window sensor does not work properly, and shows that the window is open. Shouldn't we let the user to turn the air-conditioning on?

People are used to live in non-intelligent buildings, so they generally know best what to do. If it is in contradiction with what the system wants to do, we should follow the rule, that "**User is the smartest**". A system should display warning that the weather forecast said it will be raining soon, but still let the user start lawn sprinklers. User interacts with system through upper two layers, which have lower priority than the bottom layer. **The user than has priority higher than middle layer, but still lower than bottom layer.** This can also be a good guideline for distributing tasks between bottom and middle layer.

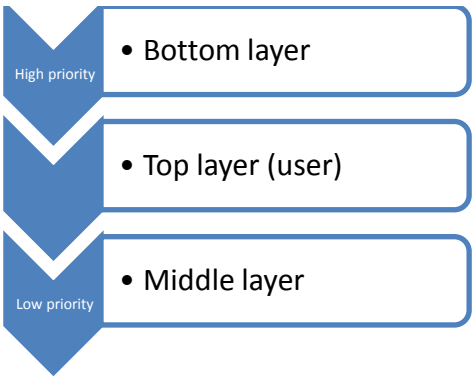


Figure 54 - Priority

3.8. Connection with top layer

In traditional two-layer control systems, the top layer is connected directly to the bottom layer. We have essentially three ways of inserting the middle layer in the system.

1) Add middle layer above the bottom layer, but leave direct connection between bottom and top layer. This way we can simply enter an existing system, but top-layer applications will not take advantage of the abstract data structures defined in the middle layer.
2) Insert middle layer in between the other two layers. The top layer has no connection with bottom layer. This solution fully takes the advantage of layered design, but middle layer also becomes the bottleneck. Creating interface so that the middle layer is transparent to the other parts of the system can be troublesome if not impossible for some existing systems. Otherwise the top layer must be able to communicate with a third-party counterpart rather than the original bottom layer.
3) A combination of the previous two. It takes advantage of both possibilities, but also can make mess in the control.

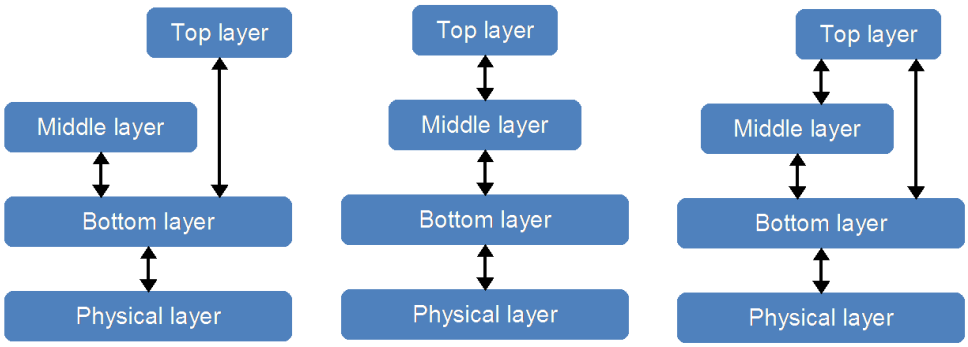


Figure 55 – Possible layer connections

It is generally possible to connect top layer application either directly to the bottom layer or through middle layer. Some top layer interfaces may require connection with the bottom layer, while others can be connected through middle layer. Because Top layer's priority is between middle and bottom layers', we can keep the priority scheme in both cases. We have to be aware of the difference amongst those two cases, though. While in the first case, action executed in top layer is performed in the bottom layer and the result appears to middle layer with bottom layer's priority. In the second case, an action coming from the top layer is first processed in middle layer and then appears to the bottom layer with middle layer's priority.

For most existing systems, we can keep the connection between top and bottom layer, but leave top layer's commands for processing in the middle layer. However difficult it sounds, it is pretty straightforward: *For example, in Figure 56 a button is pressed in the top layer, which reflects in a bottom layer variable B1. Middle layer detects the change and executes rule that in this case turns on a light. It sets corresponding bottom layer light variable L1, which reflects in the top layer and the light turns on*.

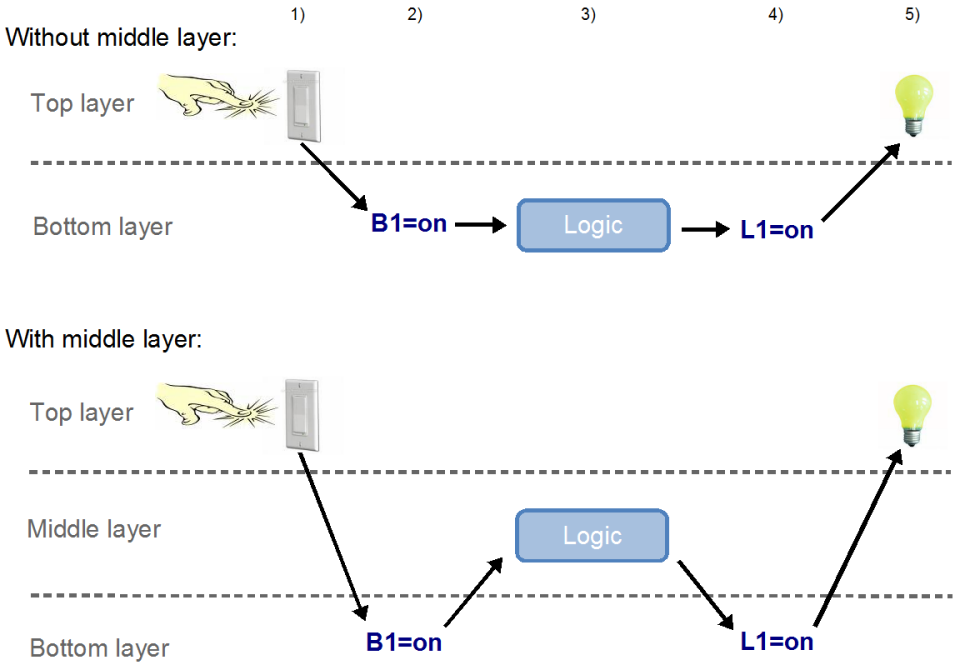


Figure 56 - Adding middle layer to existing systems

4. Hardware structure

The design of software suggested above is independent on a particular hardware implementation. The middle layer is actually suitable for bridging multiple technologies under one interface, so that we can combine for example accessible X10 switches with other providers' HVAC control and security system. In this chapter, we will look for hardware realization that would suit the needs of home automation.

4.1. Centralized versus distributed control approach

Even though distributed intelligence is now "in fashion", there is an everlasting struggle between supporters of centralized and distributed solutions. The pros and cons of both approaches are clear. Central control offers better coordination of devices and high-level functions, but acts like a bottleneck for communication and computation. If central unit fails, the system all fails. Distributed systems are less vulnerable too communication overload and one unit's failure, but each unit is much more complicated and there might be lots of redundancy. We can upgrade centralized system simply by upgrading the central unit, but it is easier to expand a distributed system.

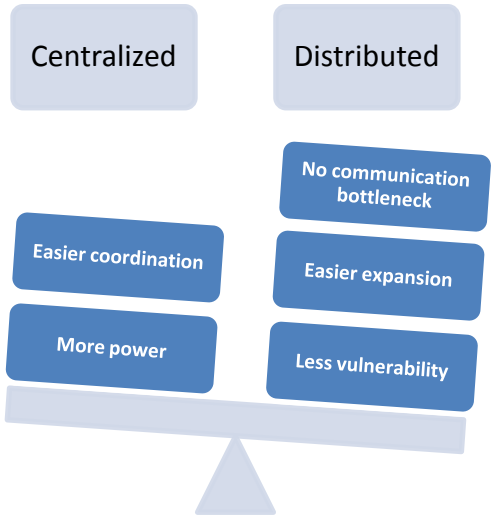


Figure 57 - Centralized versus distributed control

Most of the systems actually find a compromise somewhere in between and apply hybrid (mixed) control as a combination of both approaches. But still, what is the clue for choosing more centralized or more distributed system?

4.1.1. Inspiration in nature

We can get inspired in nature. Not considering God, The whole Earth is a great example of a distributed system with no central control. On a smaller scale, there are communities with hybrid control. Those are countries with governments or ant-hills with their queens. Going further down, there are single organisms, such as human or ant, with their brain as a typical example of centralized control. The brain does not control all the processes within organs. Using engineering terminology, for the brain an organ is a black box with inputs (hormones, nerves) and outputs (nerves).

Figure 58 - distributed, hybrid and centralized control on Earth depends on the scale

As can be seen in the nature, level of centralization varies with scale, and it seems particularly dependant on two factors:

- Central unit's processing power
- Communication capability

As long as none of the above factors limits the performance of the whole system, central control is more efficient than distributed. Can you imagine how would you walk if each of your limbs had its own little brain and just discussed with other organs what to do?

Human has been more successful than other mammals because of its brain, the powerful central unit. Communication within human body is fast enough not to impede the performance. In computer science as well as in human body, it is vision that needs the highest data transfer rate. The capacity of eye nerve data channels is just a fraction of the full image our eyes see and therefore lots of preprocessing is done in retina before the signal is sent to brain. A centralized control system concentrates all decision making, but it does not mean that its units have no own intelligence. A thermometer in a smart home with central control also does not send the current voltage and other raw values to the control center, but it already provides it with a temperature.

To be more efficient, anarchical groups of individuals in all cultures found out they needed some kind of central control and thus a government was born. It has no capacity to directly control each individual, but can coordinate effort to achieve much greater things than the individuals alone interacting amongst themselves on the same level. Communication has always been the limit for size of a country. The largest empires have always felt apart, because communication of remote parts with the

center was too slow. Only recently, when technology shrunk communication distance all over the planet to minimum, multi-continental companies and alliances are possible.

An ant hill is controlled by the ant queen using pheromones as the communication channel, which is fast enough to maintain control within one ant hill. There is no central control over all the anthills within a forest simply because ants do not have a mean of communication sufficient for this scale.

If ant's brain stops working, all the rest of the body is of no use and all the system fails. If one ant in the ant-hill dies it is replaced and the community goes on. For small, closed systems a central control may seem beneficial, while distributed control suits large systems.

Should we draw a conclusion from this inspiration by nature, it could sound: "**Central control is suitable for systems where communication and central unit's processing power do not limit the performance and where consequences of potential central unit's failure are acceptable."**

4.1.2. Application for smart homes

What is the implication of the statement mentioned above for home automation? I believe the scale of smart home control can be compared to control of a human body. With current network technologies, the communication of all the appliances with a central control unit is not a limiting factor, neither the processing power - especially if we use a PC based central control. If a central unit entirely controls a factory, the consequences of its failure are not tolerable. Both safety issues and economical losses caused by such situation are unbearable. If central control of a smart home fails, it will cause discomfort, but if a manual control option is possible, a smart home will only become an ordinary home that most of the people live in anyway. Due to frequent[15] failures of consumer electronics, customers are used to tolerate failure if fast service is provided. The biggest concern is safety, so that people do not get stuck inside a locked house during a fire, which destroyed the central control. For most of the cases a manual switch on the device (which is normally in "auto" position) will do the job.

[15] Hard drives where people now often store very valuable data have a typical failure rate of 2-4% with some models reaching up to 13% (17). Customers are accustomed to this ratio in consumer electronics, but it exceeds the limits for industrial failures by orders of magnitude.

Central control in home automation extends the horizons of functionality and decreases redundancy of intelligence in each unit, which leads to cheaper cost. As with the organs in the body, centralized should only be the control, not all the hardware. Under “central control”, some people still imagine a huge central unit with lots of extension modules and star network topology, which used to be industrial central control decades ago. On the hardware side, it is an exact opposite of what I suggest. For example, each light should have its own relay or dimmer build in (or in its proximity) and only the data channel connected with central control through a suitable network, which can be anything form a bar to a mesh wireless network. In this concept, we can keep control computer small by avoiding any relay boards and other excessive hardware, it is easier to debug problems and we can lighten the cabling, because data can also be transmitted wirelessly. In addition, switches do not need to be connected to the power line and become therefore safer and easy to install anywhere – see Figure 59.

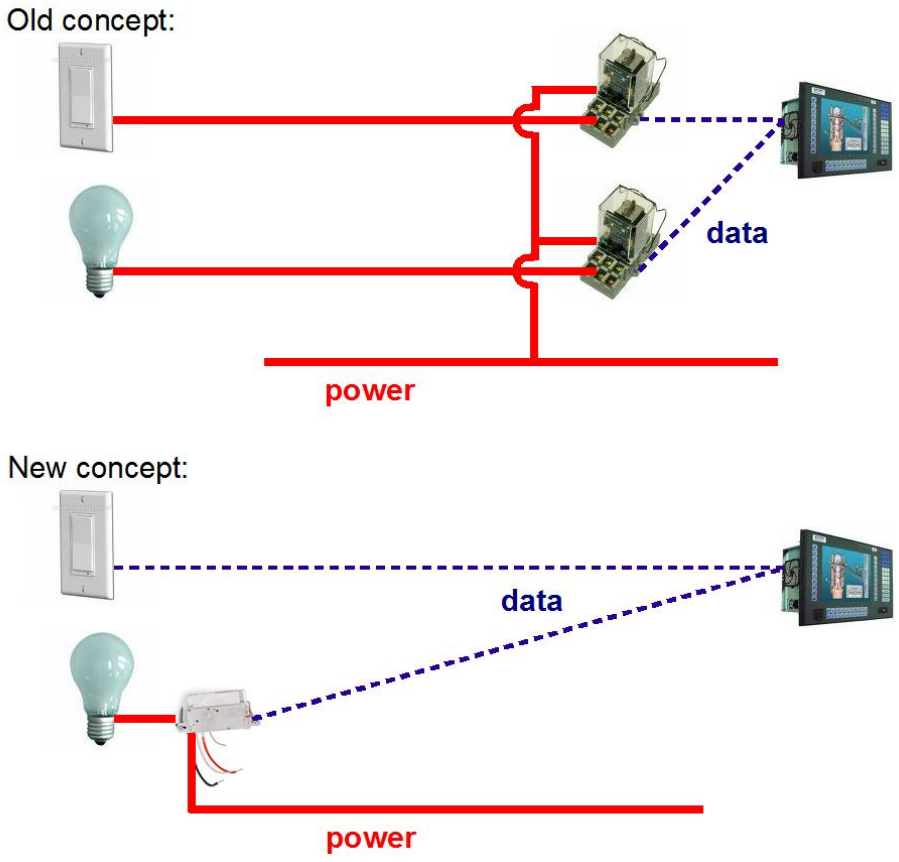


Figure 59 - Old and new concept of centralized home control

4.2. Suggested Smart Home Network structure

Until now, we have designed middle layer mostly independent on the bottom layer technologies. In this chapter, we will describe network hardware structure that will take the most advantage of three layer concept in a real application

There are generally two main data networks required in a smart home:

- Control Network – secure and reliable network preferably with a guaranteed response time. It is used for core home automation devices such as a switch. There are typically infrequent small packets of data sent through the network. A message could be: “button C23 pressed”, or “set D04 to 0”.
- User Ethernet – fast network connected to the Internet for large quantities of data including streaming audio and video.

We can set aside prejudice given by current systems and try to design a network structure that would be optimal for smart home and its specific needs. Figure 60 introduces one possibility.

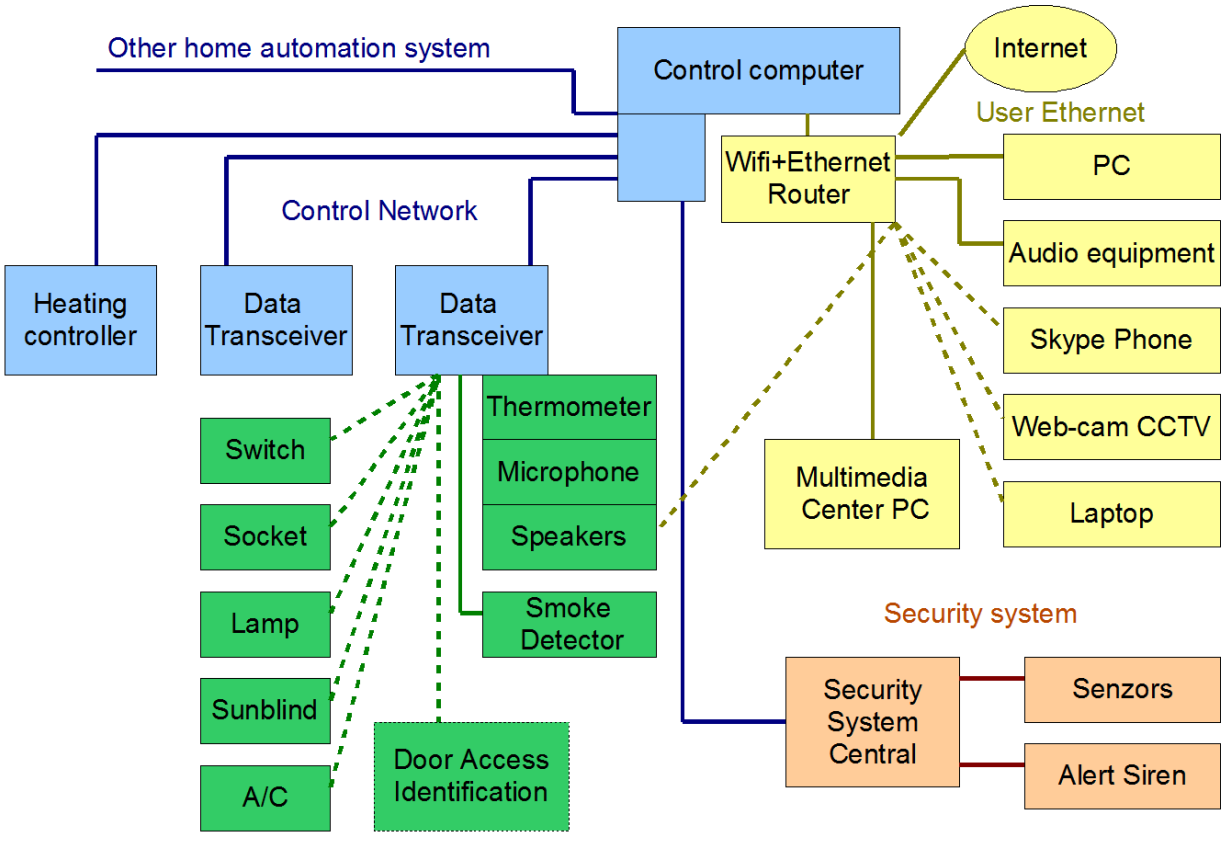


Figure 60 - Suggested smart home network structure

4.2.1. Control Computer

Control Computer is the brain of this control system, where most of the intelligence comes from. It is a machine based on reliable PC components that guarantee high performance with low price and great compatibility. Even though it could also be an ordinary PC, we can customize the machine to suit home automation requirements, which differ from requirements on a personal computer. The central computer is supposed to operate 24/7 for many years (nobody wants to replace control system that is already debugged after period typical for replacing a PC). We will focus on brand components that guarantee long operation time and low power consumption. Most of the malfunction occurs in moving parts, so we can use FLASH as the main memory, where system is installed, extensible with memory cards or a hard-drive to store non-vital data and backups. We also had better to employ only passive cooling. Actual processor speed and graphic card are not as important, but the modern multi-core processors are suitable for multi-thread control SW, that can guarantee fast response. The computer can all fit in a fairly small case to be stored at a safe location (typically a utility room). We can also consider using industrial PC or SoftPLC that have small and robust design, but worse performance, upgradeability and higher cost.

As for the operating system, we need stability at the first place. Since user will not use this computer for other purpose, we can afford to install system that end-users are not very familiar with, as long as the interface (normally via remote access) is very intuitive. When choosing the OS, we should also consider requirements of other control SW used in the installation. Even though it might not be the most suitable system, most of current automation software still uses Windows platforms.

4.2.2. Data Transceiver

Data Transceiver is a piece of hardware that enables Control Computer interaction with some particular network and integrates basic home automation functions. It has reliable connection with the Control Computer (preferably via cable), such as Ethernet that is independent on the User Ethernet or USB. It also has interchangeable module for connection to a particular home automation network or networks, whether it is wireless technology or a bus.

There must be at least one Data Transceiver in the system and it may be enough for small installations. If more control networks are present, there can be a transceiver for each of them. For better performance and large homes, it is better to place one Data Transceiver into every major room – typically at the place of switches by the door. We will call the space covered by a transceiver as a *zone*.

Data Transceivers have open architecture so they can be produced by many manufacturers. They are expected to be offered in a couple variations, ranging from a simple one that only connects a particular network to the computer (which actually already exists for all the networks), to well equipped transceiver, which integrates many functions and controls. It is always much more economical to integrate more functions into one device, than having a separate device for each of them.

Fully equipped transceiver includes:

- **Interface for connection to a PC**
- **Interface module for connection to particular home automation control network** (this module can be selected according to the network actually used in the installation – for example EIB, X10 or ZigBee)
- **Smoke detector** on an extension cable (smoke detector should generally be placed near the ceiling)
- **Gas detector** on an extension cable (gas detector should generally be placed near the ground and close to potential source of the gas)
- **Thermometer**
- **Thermostat controls and display** to set and view temperature in the current zone
- **Programmable buttons module** can be used for controlling selected devices in the zone
- **Touch screen** as a more luxurious, yet increasingly affordable substitution for programmable buttons and thermostat controls and display. It can also serve as an emergency signal light.
- **Microphone** for voice commands, telephony, safety (system can recognize a person calling "HELP") and security (can report a sound detected when nobody is supposed to be present)
- **Speaker** for voice feedback, telephony, alarm sound and possibly even music (given small size of the transceiver, it will not be good enough for quality audio)
- **Backup battery** to maintain functionality even during power blackout
- **"Manual control" switch** that can make transceiver use settings, which have been set as default, and prevent receiving commands from the Control Computer. This can be used in case of Control Computer's failure.

Even though Data Transceiver does not have much intelligence by itself, it can implement some basic safety instructions, such as playing alarm sound when smoke detector is excited. Control network's bandwidth should be sufficient for audio stream transfer from the microphone and to the speaker in most of the cases. For slow control networks, the audio may also be transmitted over the User Ethernet.

4.2.3. Control network

There are many networks currently used for home automation and they can almost all be used with this system. Wireless networks designed for home automation are becoming cheap and reliable enough to replace current bus based wired networks. Insteon, ZigBee, and Z-Wave are relatively new technologies that use the mesh network configuration - all the devices within a network are equipped with radios which can both receive and transmit signal and therefore work as repeaters to extend the network's range. They seem to be about to replace the old X10 and other networks (Insteon is backward compatible with X10). Radios use new power-efficient technology and add just about $20 to the price of any device that is about to be controlled – light, switch, Data Transceiver…

Control Computer is then used to configure the network and perform higher intelligence tasks.

We can also consider a technology that would get the most out of Data Transceivers. If we place Data Transceiver to every main room, there will be sufficient network coverage for all the wireless devices. Therefore each network unit only needs very simple communication chip. Instead of communicating all together like in a mesh network, each slave unit only communicates with its Data Transceiver master. A master-slave model is a very reliable network with simple and cheap units. When a button is pressed, the message is first translated via Data Transceiver to the Control Computer which, according to the defined rules, decides what to do. For a typical rule at Figure 61 that says: "*when button B1 is pressed, turn light L1 to 100%*" the button will use Data Transceiver to pass the information to Control Computer. It decides to turn on L1 and passes order back to the network though Data Transceiver. In this case we have full control over all the units, but the journey such simple request has to undergo is too long, so it can be slow and vulnerable to unit failures.

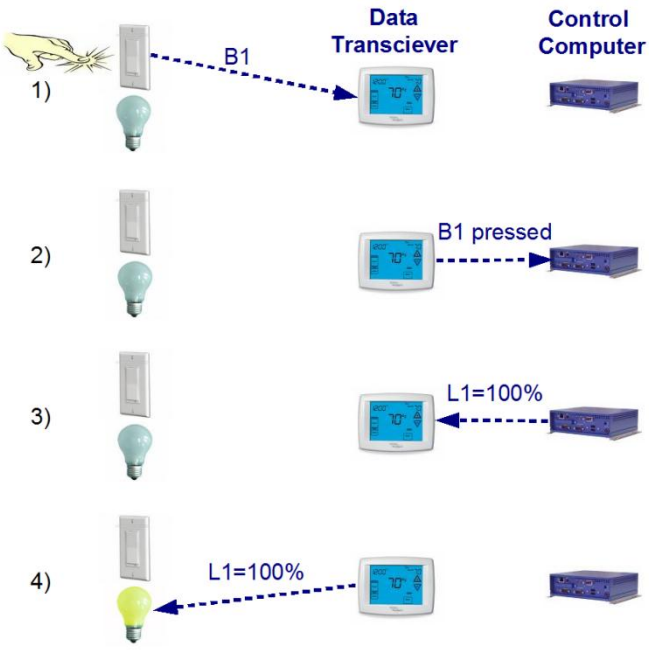


Figure 61 - Control without command register

We can improve this behavior by equipping Data Transceiver by a *command register* – a simple memory, essentially a table which says what signal to send in the control network immediately in response to receiving particular signal. The definition of this table is downloaded to Data Transceiver from Control Computer according to current logic rules. This way as soon as signal that button B1 has been pressed reaches Data transceiver, it finds table entry for B1 to set L1 to 100% and it immediately sends order to L1. Data Transceiver also informs Control Computer that B1 has been pressed. Control Computer can log the event and eventually process other actions caused by pressing B1 in a normal way.

IF	THEN
B1	L1=100%

Table 4 - Command register example

If there is a rule, that pressing B1 will set light only to 70% between 11pm and 5am, Control Computer will simply adjust the Data Transceiver's command register at 11pm and set back at 5am. If we wanted to achieve this function without central control, each distributed unit would have to have own clock, means of synchronization, program memory and so on, which would make all the units much more expensive.

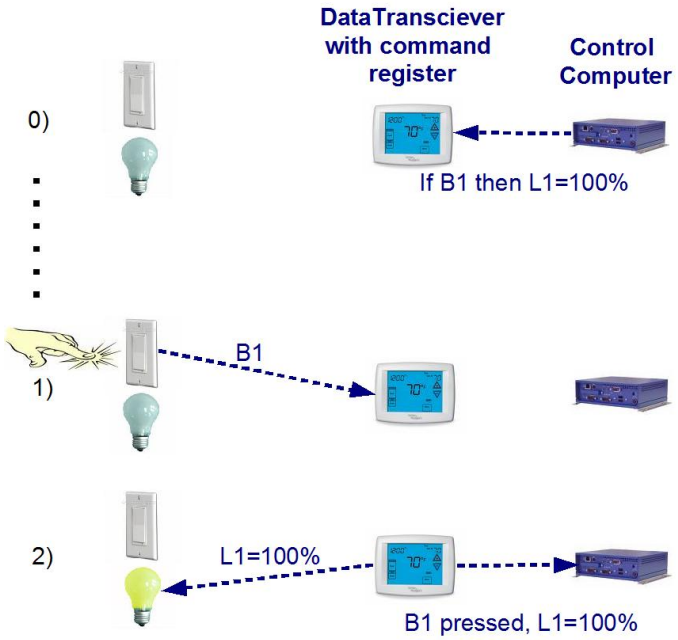


Figure 62 - Control with command register

By employing command register, we reduce the response time (it is important that a light turns on immediately when a switch is pressed) and also reduce the consequences of Central Computer's failure. Even if the Central Computer for any reason does not respond, we will be still able to turn the light on. A "manual control" switch can simply make the transceiver use a default command register.

Using command register is completely transparent for the logic rules and will only apply to simple rules within the same zone. In case we control devices in one zone from another zone, it is usually not a vital control and we do can tolerate slightly higher response time or failure in case of Central Computer's malfunction.

Implicit rules that are introduced in 3.3.5 provide direct control between devices without the central element in a way that is used in most of the current networks. As can be seen at Figure 63, Control Computer configures network according to the rules. Implicit rules are then executed directly within the network devices and Control Computer is only informed about the actions. This approach is most stable and should be used for all the implicit rules and other simple rules that can be directly translated into the network protocol. It is again completely transparent to the user in logic definition.

Most of the logic in home automation is as simple as “this button controls this device” and therefore executed as implicit rules. However, more complicated logic that cannot be translated into the network devices is still processed by the Control Computer in one of the fashions described above.

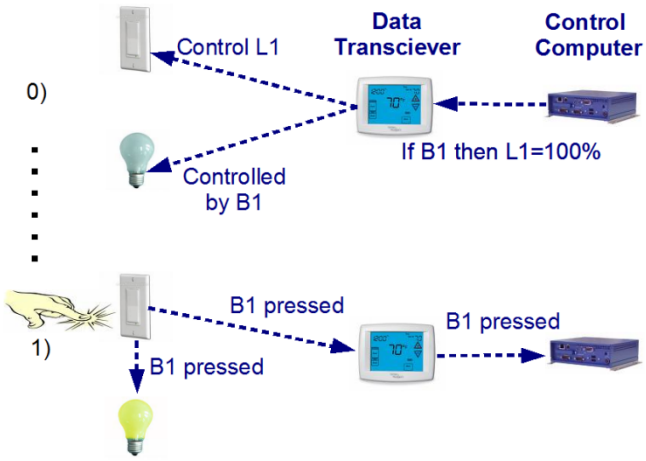


Figure 63 - Executing Implicit Rules

4.2.4. Security System

Security systems are usually separate from home automation systems, because in many countries a significant discount on insurance is provided for homes using licensed security system. Smart home automation networks are too open and unsecure to obtain this license. Some security systems, however, can be connected to the home automation system, at least in a form of read-only information from motion and other sensors.

4.2.5. User Ethernet

User Ethernet is a high-bandwidth home network with access to the Internet mostly for home entertainment. It usually employs high-speed Ethernet with Wi-Fi router. It is used to connect all the computers within the household to the Internet and transmit high volumes of data such as audio or video streams.

A nice example of a device fully using the User Ethernet is Media center computer which integrates all the multimedia entertainment such as television, movies, music, photos into one device. It offers even more entertainment with permanent connection to the Internet. Media center is usually controlled by a simple remote control and can serve as a great top-layer interface of home automation. Unlike the Control Computer, which is not directly accessed by the user and only little upgraded over years, Media Center is a high-tech PC with huge storage capacity that will be upgraded often to maintain best performance required by multimedia applications. User can directly access the system and use Media Center as an ordinary PC for office applications.

It can for example record evening news and users may later view them streamed on their laptop, a simple media extender added to any television will bring all the content of Media Center to other device.

Smart home can also be equipped with a simple CCTV security monitoring system realized by web cameras transmitting over the User Ethernet. This is much cheaper alternative of an ordinary analog CCTV system and it suits the requirements of home usage even better. One of the biggest advantages is that we do not have to install any additional wires for Ethernet webcam monitoring system and we already get a digital signal.

Internet telephony is becoming increasingly popular and some households already use it instead of an ordinary land line. With a growing number of appliances such as the Skype phone, we will be soon able to replace all the ordinary telephones at home with phones connected to the User Ethernet and achieve more comfort with lower telephone bills.

5. Further ideas for home automation

Employing middle layer and the control computer described in previous chapters opens almost endless possibilities for advanced home automation features that will once make our living more enjoyable, yet safer, cheaper and ecological. A few examples are provided in this chapter.

5.1. Power consumption optimization

It is a general expectation that smart home will consume resources wisely and thus become both economical and environmental friendly investment. Automated home does not save energy by itself, it is again matter of a smart control software. There are several ways of decreasing energy consumption:

5.1.1. Optimal heating settings

Heating and cooling costs depend on climate, heat transfer dynamics of the house and targeted temperature, but generally stand out as the highest running cost of a house (typically account for 50 - 80% of energy costs)

The key point is to use heating only where it is necessary and when it is necessary. Whilst in winter comfortable temperature in the living room is around 22°C, a temperature suitable for healthy sleeping is 18°C. In places such as a corridor, 15°C is usually sufficient. Temperature in the house can be decreased to around 17°C when nobody is present or even lower for long periods such as a vacation. Similar scheme can be found for summer cooling. Not many people realize that reducing heating by 1°C saves about 5% of the energy. Nobody likes to wake up to a cold room or come into cold house freezing from outside. Here comes the intelligent home, which knows when its inhabitants wake up, leave the house and come back, so that it can take care of optimal heating. Much more efficient heating control can be achieved by smart PC software, than using a thermostat. The software can analyze historical data to identify how long it takes to heat up the house and therefore start the heating in an optimal moment before people come back. It can also take advantage of weather forecast or provide the user with heating cost analysis based on real consumption data and execute a wizard for reducing the expenses. The user can than make informed decision about setting the comfort/economy ratio.

Many tips for heating savings can be found online, such as (10)

5.1.2. Maximizing benefits from external factors

By connection to the Internet and access to many sensors around the house, a smart home has much information that it can use to be more efficient.

Local weather forecast can be invaluable for heating / cooling control. We can accumulate cold at night by automatically opening ventilation when outside temperature becomes lower than our set point, or accumulate heat during the day in a similar manner.

Weather forecast and rain sensors can also contribute to intelligent lawn watering, so that the grass does not die during dry days and the sprinklers do not waste water when rain is coming.

Knowing exact date, time and geographical location and layout of the house, Control Computer can do simple math to compute position of the sun and continuously adjust blinds angle accordingly for

either maximum lighting or shading, so that the blinds either reflect the light outside, or let as much of it as possible inside. The full power of sunlight on the surface of Earth is about 1.4 kW/m^2 (11). It depends on location, season and weather and time of the day, so for simplification at 40 degrees of latitude we can expect an average power of 600 W/m^2 during a summer day. Which means 8 hours x 600 W/m^2 = 4.8 kW/m^2 per day (12). Multiplied by the surface of exposed windows in the house, we will find that sunlight is a great power that can significantly contribute to heating / cooling when used wisely.

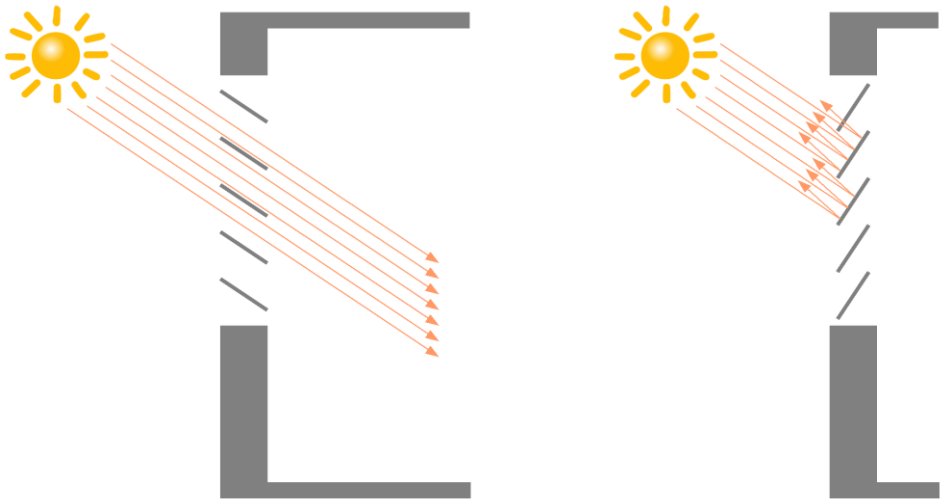


Figure 64 - Adjusting angle of the sunblinds for maximal heating and shading

Adjusting the angle of sun-blinds affects both heating and lighting. There are already controllers such as The Dutch installation (13) that take both effects in account. Those algorithms can also run with advantage on the Control Computer and add significant intelligence and savings to ordinary controllable sun-blinds.

5.1.3. Reducing device usage

The Control Computer has information when all the controlled devices are running. We can let the user to add power consumption [Watts / hour] parameter to each device. System can then track their usage and simply identify most power-consuming devices, inform user about devices that are running even when nobody is present. Instant calculation can show how much energy cost the user could save on those devices per year.

We can also collect real-time power consumption of the house by a simple circuit and track and display how much we are currently spending for energy. There are also stand-alone devices for this purpose such as the PowerCost Monitor.

Figure 65 - PowerCost Monitor

5.2. Simulation mode – presence simulation

Many home automation systems require special settings for presence simulation to specify when exactly a particular device turns on and off during the simulation. System then becomes more complicated and predefined simulation often does not look as natural as when somebody is really present. (*For example Christmas lights turned by timer in Home Alone movie*)

This is completely unnecessary with Central Computer structure, because all the events are logged and stored at the Central Computer in FIFO manner (which means that if memory fills, the oldest logs are removed). The memory should however be big enough to log all the information for at least the past month, but preferably even much longer. When the house is turned into the simulation mode, it simply takes lighting events for the last week and executes them in the same manner, or even better with little random delay. The house than looks exactly like when somebody was present, but requires no user settings.

5.3. Security web-cameras

We have already touched usage of web-cameras for security monitoring in Chapter 0. Web-cameras compared to traditional CCTV systems are cheaper, can be connected directly to the user Ethernet network and provide digital image of sufficient quality.

When the door-bell rings, we can display image from a camera placed by the entrance at any chosen devices, including touch screen displays of Data Transceiver, Personal computer or a television connected to Media Center Computer. In addition, we can attach stills of the nearby cameras to alarms and events. We can therefore capture image of a burglar that has been captured by outside camera and instantly send via email or instant messaging to a responsible person, before the thief even has chance to destroy the camera or other smart home devices.

Image processing needs lots of computational time, but already with current hardware, we can also afford to monitor image from the cameras and detect big changes against normal. Those changes can also be used to fire events/alarms with particular image included.

6. Summary

In the thesis I have analyzed current smart home control systems and home automation market situation. I identified impropriate control software to be the main cause why home automation is not as common and powerful as the hardware allows.

Taking differences of various home automation users into account, I suggested a three-layer control system would have many advantages over commonly used two-layer control systems. I provided detailed specification of the new middle layer software, which uses agents to represent variables and simplified system of logic rules. I have created examples of logic function blocks, clarified processing of rules and their execution in bottom layers. I have shown a pilot middle layer interface, that represents smart home control in four main views: home, scheme, logic and table. Split screens, basic/advanced views, colors and shapes are employed to create powerful, yet intuitive interface. I have introduced alarm management, priorities and guidelines for connecting middle layer into existing systems.

Then I suggested suitable hardware and network structure that is based on a combination of powerful central control with robust distributed execution.

Finally, I provided some tips to make smart homes smarter and more economical. I hope my innovative ideas will contribute to moving home automation further.

The mayor challenge for developers of the middle layer system will be communication drivers ensuring flawless connection to other systems, which includes exchange of variables, alarms, downloading logic, but also plug and play connectivity of new devices. I suggest open architecture so that even producers of smaller systems can write communication drivers for their own devices.

Bibliography

1. **Papageorgiou, M.** *Multilayer control system design applied to freeway traffic.* 1984. pp. 482- 490. 0018-9286.

2. **Gat, Erann.** On Three-Layer Architectures. [Online] Appears in Artificial Intelligence and Mobile Robots. http://www.flownet.com/gat/papers/tla.pdf.

3. **Thomas, Philip.** Three-Layer Architectures. [Online] 2006. http://dora.eeap.cwru.edu/msb/591/3layer.pdf.

4. **IEC.** Programmable controllers - Part 3: Programming languages . [Online] http://webstore.iec.ch/webstore/webstore.nsf/artnum/029664.

5. Basic Electricity Tutorial. *1728 Software Systems.* [Online] [Cited: 11 24, 2007.] http://www.1728.com/project2.htm.

6. OPC Alarms and Events Overview. [Online] 3 2007. www.easydeltav.com/pd/FWP_OPC_Alarms_Events.pdf.

7. **Health and Safety Executive, UK.** Better alarm handling. [Online] http://www.hse.gov.uk/pubns/chis6.pdf.

8. **Norwegian Petroleum Directorate.** Principles for alarm system design. [Online] 2001. http://www.ptil.no/regelverk/R2002/ALARM_SYSTEM_DESIGN_E.HTM.

9. **Randell, Brian.** Concurrent Exception Handling and Resolution. [Online] 2002. http://homepages.cs.ncl.ac.uk/alexander.romanovsky/home.formal/award2002.pdf.

10. Cutting Home Heating Bills. *Chiff.com.* [Online] [Cited: 11 13, 2007.] http://www.chiff.com/a/cut-heat-bills.htm.

11. **Elert, Glenn.** Power Density of Solar Radiation. *The Physics Factbook.* [Online] 1998. http://hypertextbook.com/facts/1998/ManicaPiputbundit.shtml.

12. Basics of Solar Energy. [Online] University of Oregon, 1998. http://zebu.uoregon.edu/1998/ph162/l4.html.

13. One system - one integrated solution. [Online] 10 25, 2005. [Cited: 11 13, 2007.] http://www.bsee.co.uk/news/fullstory.php/aid/3512/One_system_-_one_integrated_solution.html.

14. Bibliografické citace. *Knihovny CVUT.* [Online] [Cited: July 10, 2007.] http://knihovny.cvut.cz/vychova/vyuka-fs/diplomka.html.

15. *Projekt Bibliografické citace.* [Online] [Cited: July 10, 2007.] http://www.citace.com/.

16. **Rodriguez, Marco A.** Guidelines for control software organization and development. [Online] ISA, 2003. http://www.isa.org/InTechTemplate.cfm?Section=InTech&template=/ContentManagement/ContentDisplay.cfm&ContentID=31247.

17. **Myllymaki, Matti.** Control system for building automation controlled by human physiological signals. *Patent Storm.* [Online] 2002. http://www.patentstorm.us/patents/6348867-fulltext.html.

18. Building Automation and Control System. [Online] The Univerity of New South Wales, 2005. http://notessrv.chan.unsw.edu.au/Faciliti.nsf/pages/man/$FILE/app_1.pdf.

19. **Kikta, Christopher, et al.** Small building automation control system. [Online] 2002. http://www.freepatentsonline.com/20020152298.html.

20. **Elmahdy, A.H.** An Overview of Central Control and Monitoring Systems. *Canadian Building Digests.* [Online] 1981. http://irc.nrc-cnrc.gc.ca/pubs/cbd/cbd214_e.html.

21. **Technologies, JDS.** Manual for all STARGATE-IP, STARGATE-Lite, and CS30 Controllers. [Online] 2000. http://www.jdstechnologies.com/download/stargate/sgmanual.pdf.

22. Disk drive failures 15 times what vendors say. *Computerworld.* [Online] 02 03, 2007. [Cited: 11 1, 2007.] http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9012066.

23. INSTEON Compared. *SmartLabs, Inc.* [Online] 01 02, 2006. http://www.smartlabsinc.com/files/INSTEONCompared20060102a.pdf.

Additional resources:

www.wpclipart.com and www.iclipart.com - clipart galleries

Appendix A – Overview of specialized home automation SW

There are many programs designed especially for home automation. Hardware developers usually provide software for their own system, but there are also third party applications, which often try to bridge several technologies. Here is a list of some of those applications with screenshots of their GUI. It is not a complete enumeration of all the software available, but it can help to make a better picture about current state of the industry.

ActiveHome

www.x10.com

Very beautiful and intuitive home automation software developed by X10 that makes switch logic, timing functions and web-cameras easily accessible to everyone. It allows basic macros, but it does not integrate HVAC support and more complex tasks are impossible to realize.

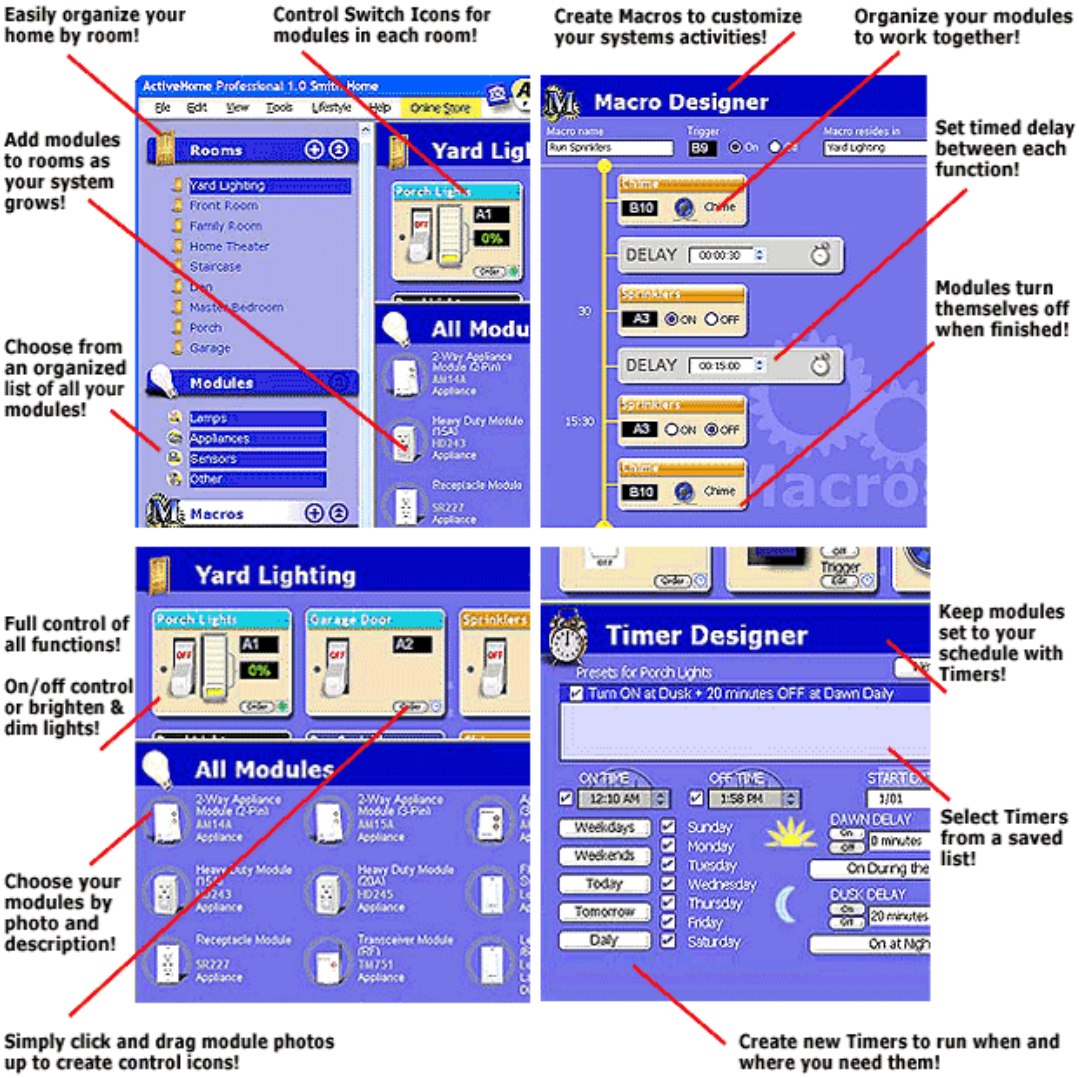


Figure 66 - ActiveHome features overview

AutomateThis!

http://www.homeautomationdev.com/

AutomateThis! is a table-based home automation status viewer, logger, and control program currently under development. It supports HAI Omni and X10.

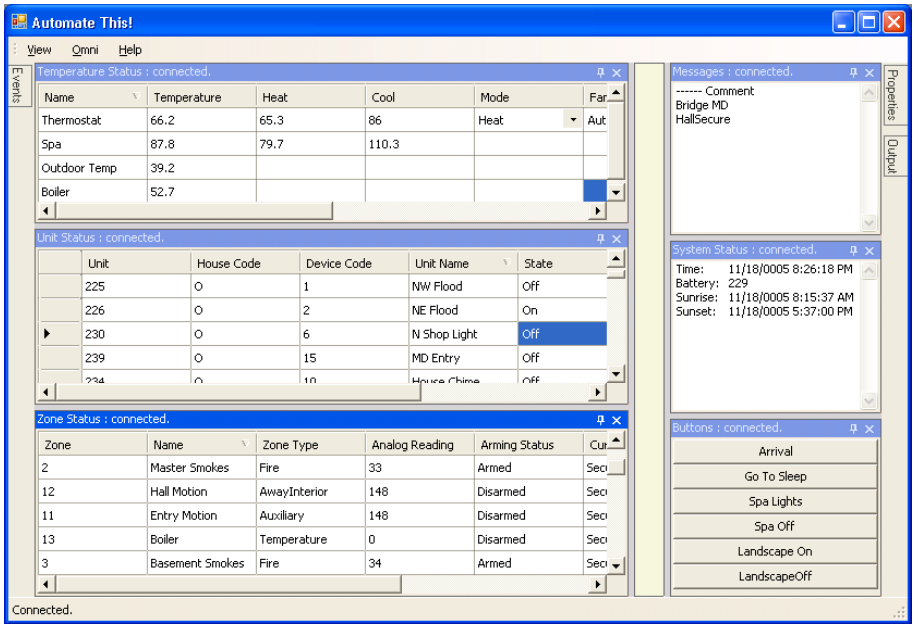


Figure 67 - Automate This! Interface

Automize

www.hiteksoftware.com

Automize is rather technical multi-platform task scheduling and automation software.

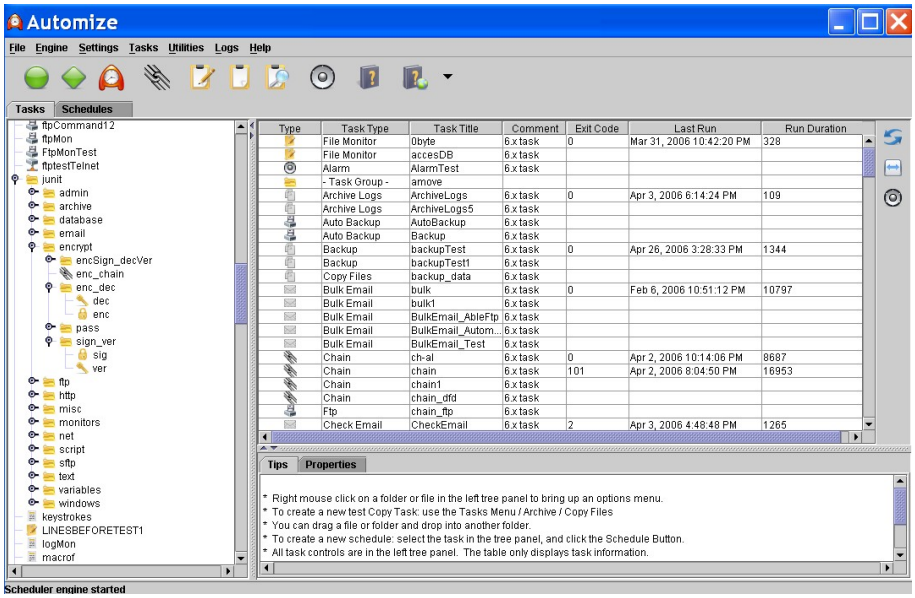


Figure 68 - Automize - Task scheduler

ConfigNet

www.supernasystems.com

Superna ConfigNet™ is desktop software that facilitates the creation, maintenance and management of Digital Home solution. It consists of Project Manager to manage devices, Automation Manager to schedule tasks and Screen Builder to design touch screen interface.

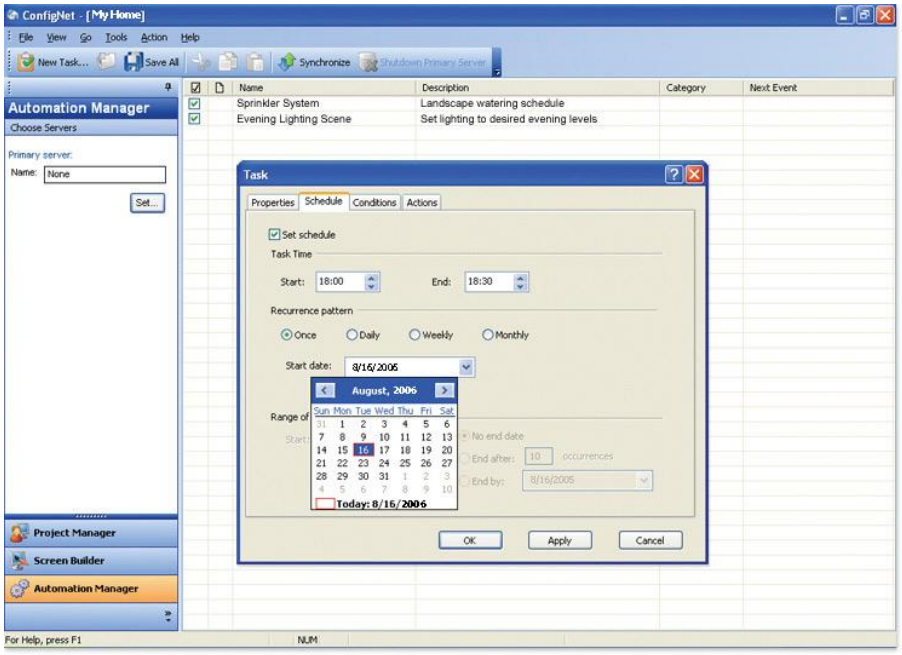


Figure 69 - ConfigNet Automation Manager

Figure 70 - ConfigNet Screen Builder

Control4

www.control4.com

Control4 Composer Home Edition is used to manage Control4 system including multimedia. Touch screens are widely used for local control, while 4Sight Web Navigator allows remote access to the system.

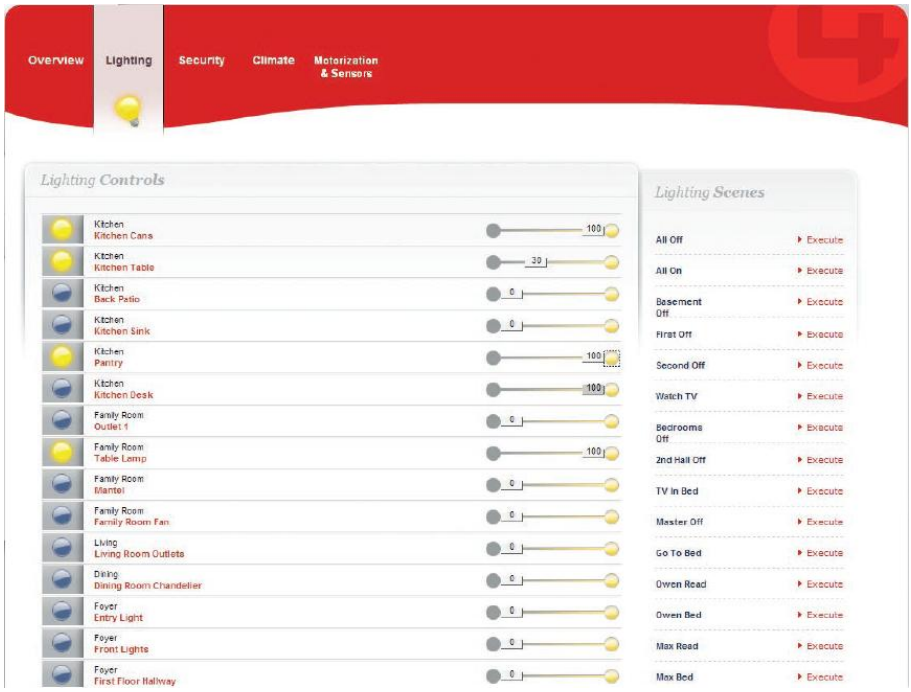


Figure 71 - 4Sight Web Navigator - Lighting Control

Figure 72 - Control4 wireless touch screen

Creston

www.crestron.com

Creston's RoomView software focuses on control of audio-video resources within a building. D3 Pro is a specialized residential lighting system, but does not limit to lighting.

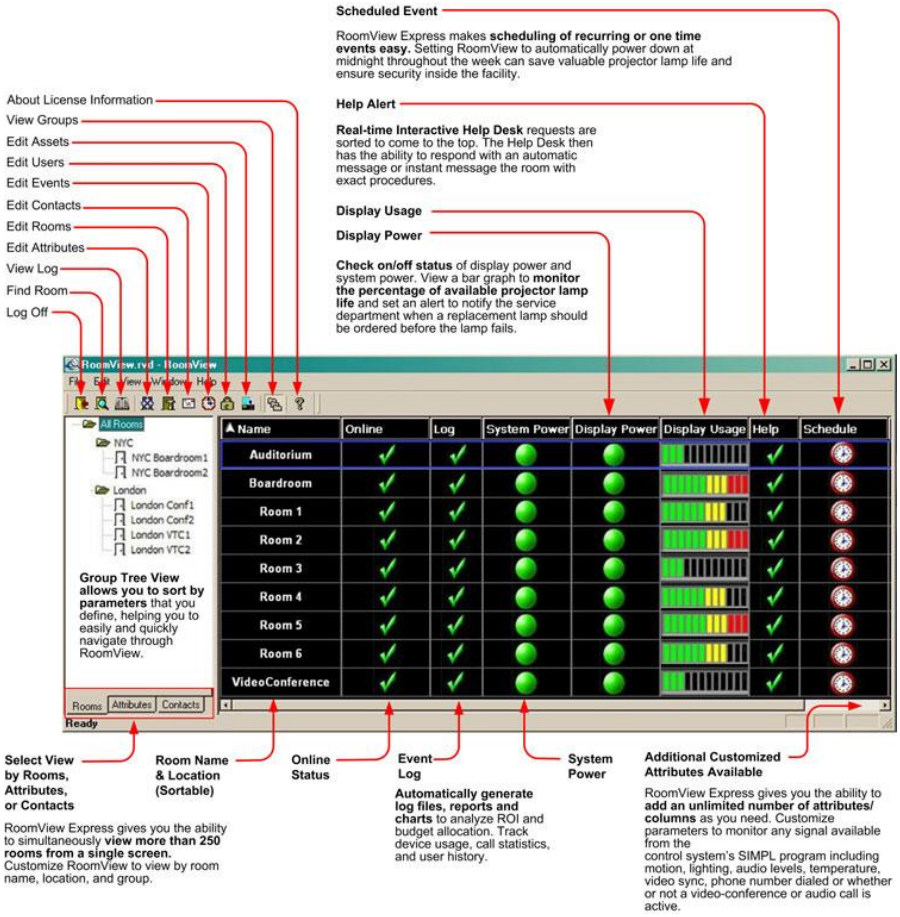


Figure 73 - Creston RoomView

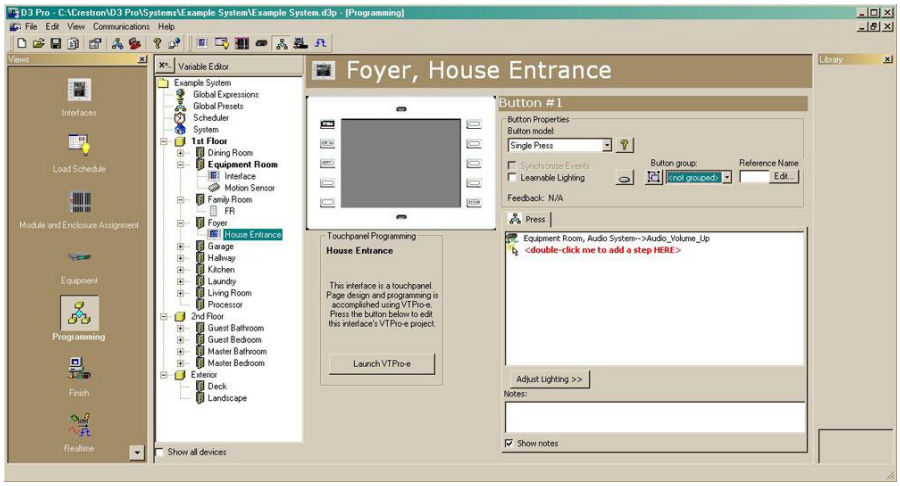


Figure 74 - Creston D3 Pro programming

HAI

www.homeauto.com

In addition to its OmniTouch touch screen, HAI offers control of UPB (Universal Powerline Bus) devices via web and Windows Media Center.

Figure 75 - HAI Web interface

Figure 76 - HAI Media Center Interface

HAL

www.automatedliving.com

Home Automated Living provides PCI extension card for PC and software to create a voice interface for control of the house. System receives orders by microphone and informs user through the audio system.

Harmony 5

www.idomus.co.uk

A PC-based command center for various systems of automated home. It includes macro language similar to Visual Basic For Applications (VBA) and Media Center plug-in.

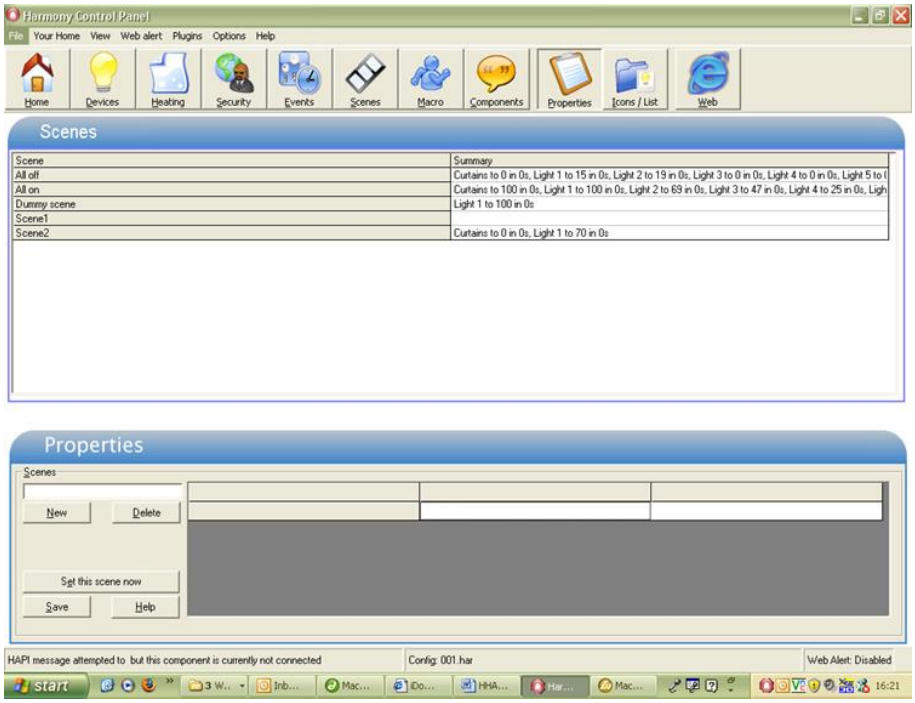


Figure 77 - Scenes in Harmony 5

Figure 78 - Harmony Media Center plug-in

HomeSeer

www.homeseer.com

Popular home automation software supports many technologies and features web access, touch screen and Media Center interface and also voice recognition. It provides a central control over various networks, but requires the PC server running at all the times. User community shares numerous VBscript, JavaScript and PerlScript scripts that can enhance HomeSeer's features.

HomeSeer's Media Center interface is a nice example of a top layer control, while the web-server is a textual application standing in between the bottom and middle layers as described in this thesis.

Figure 79 - HomeSeer Media Center interface

Figure 80 - HomeSeer Web Control interface

HomeVision

www.csi3.com

Custom Solutions, Inc. develops a web / touch screen interface and also direct TV interface for its hardware home automation controller called HomeVision.

Figure 81 - HomeVision web / touch screen control

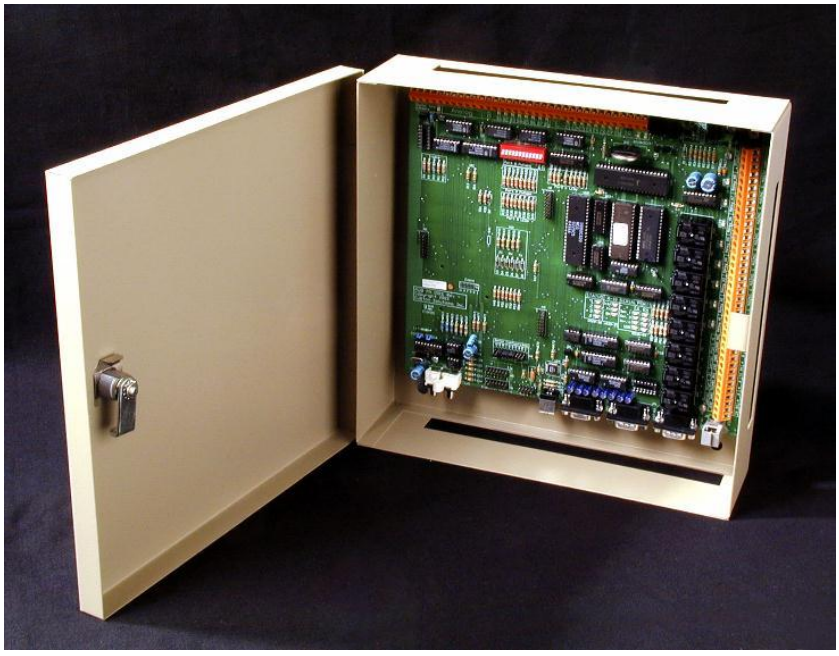


Figure 82 - HomeVision Pro

Indigo

www.perceptiveautomation.com

Home automation software providing X10 and Insteon control under MAC OS.

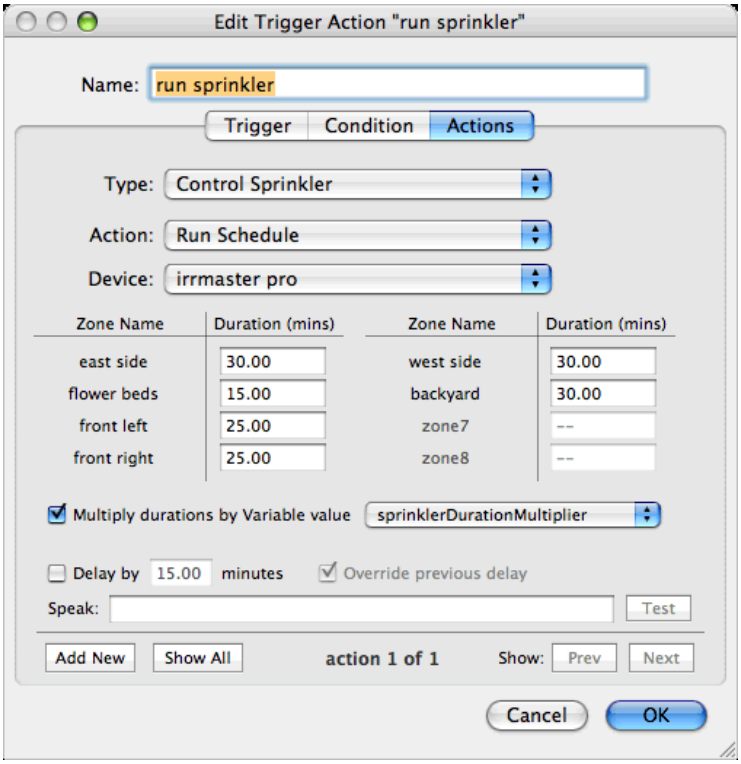


Figure 83 - Indigo Actions

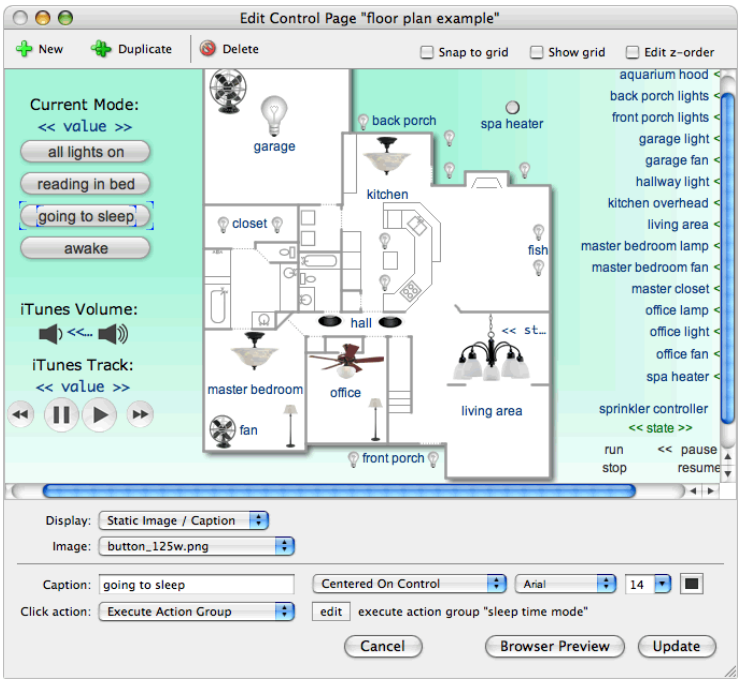


Figure 84 - Indigo Floor plan

mControl

www.embeddedautomation.com

Embedded Automation's mHome project is a solution for digital homes that focuses on home control by Media Center, touch screen, Windows Mobile or web browser applications.

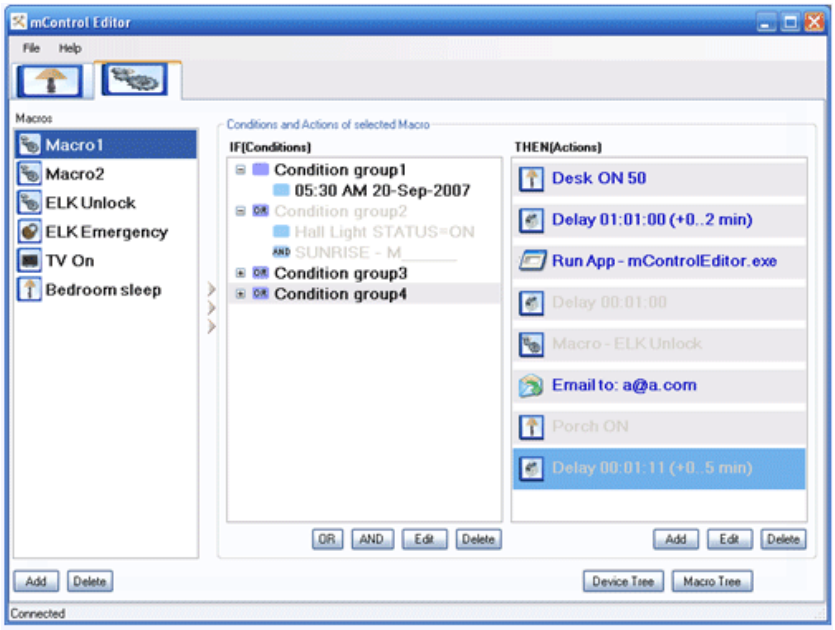


Figure 85 - mControl - Logic rules based on IF...THEN conditions

Figure 86 - mControl Media Center interface

PowerHome

www.power-home.com

A home automation software featuring Infrared, X10, and Insteon control. It is suitable for advanced users, who like to play with home automation and seek advanced features.

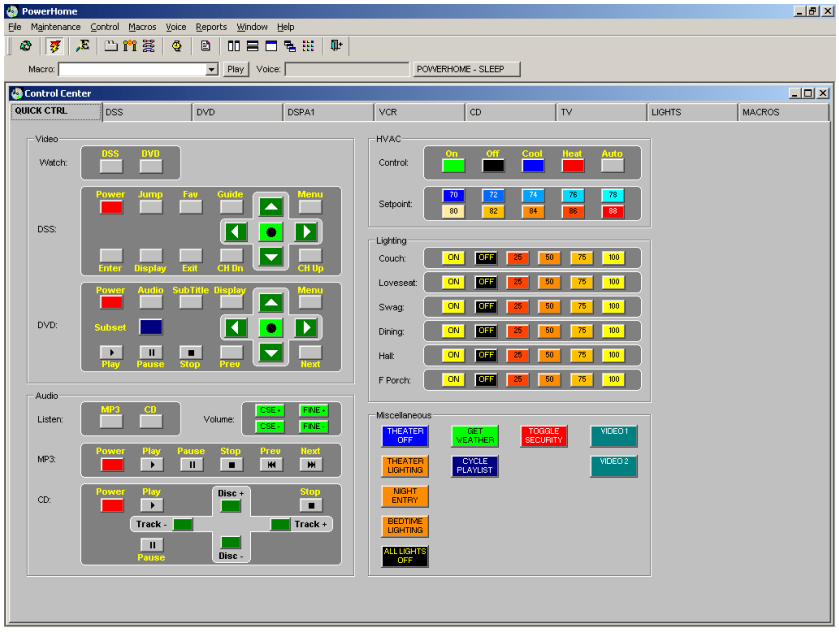


Figure 87 - PowerHome Control Center

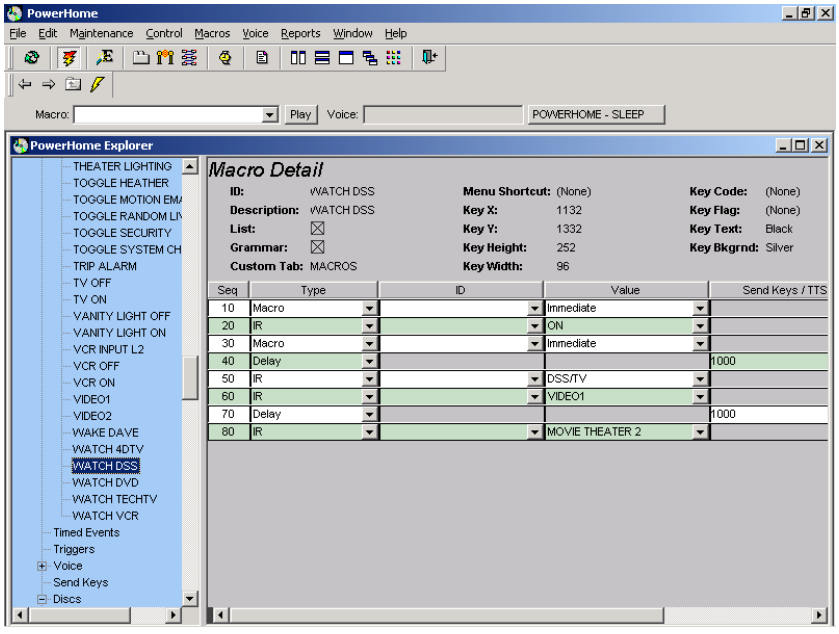


Figure 88 - PowerHome Macro Detail Explorer

Plato™

One of the first specialized visual home automation SW, but already obsolete for a long time.

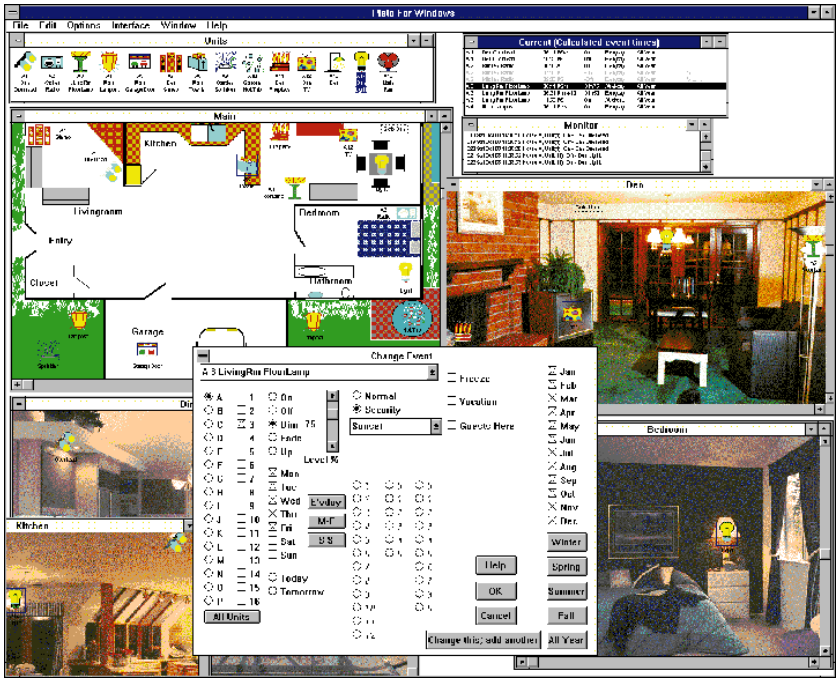


Figure 89 - Plato™

Smarthome Manager

www.smarthome.com

Smarthome Manager is a configuration SW for Powerlinc™ Controller that combines both X10 and Insteon networks.

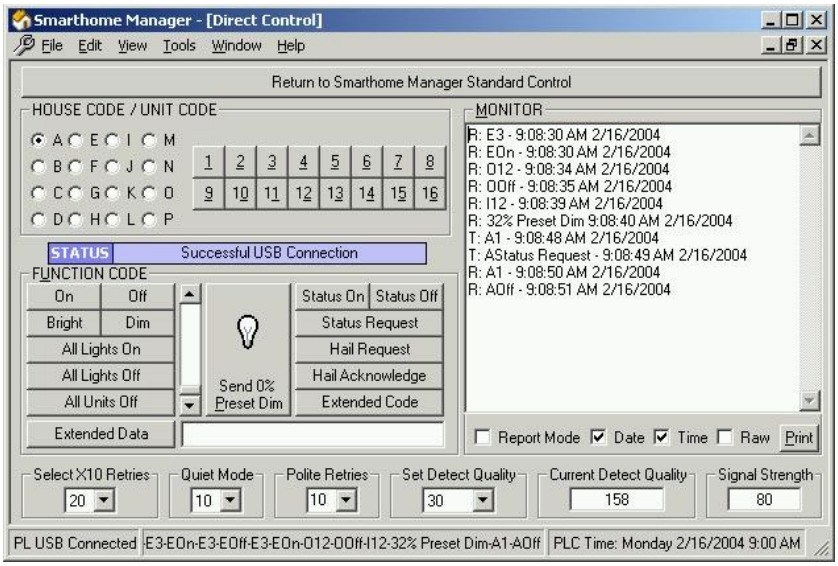


Figure 90 - Smarthome Manager Interface

UPStart

www.simply-automated.com

UPStart is a configuration tool for Universal Powerline Bus (UPB™) devices.

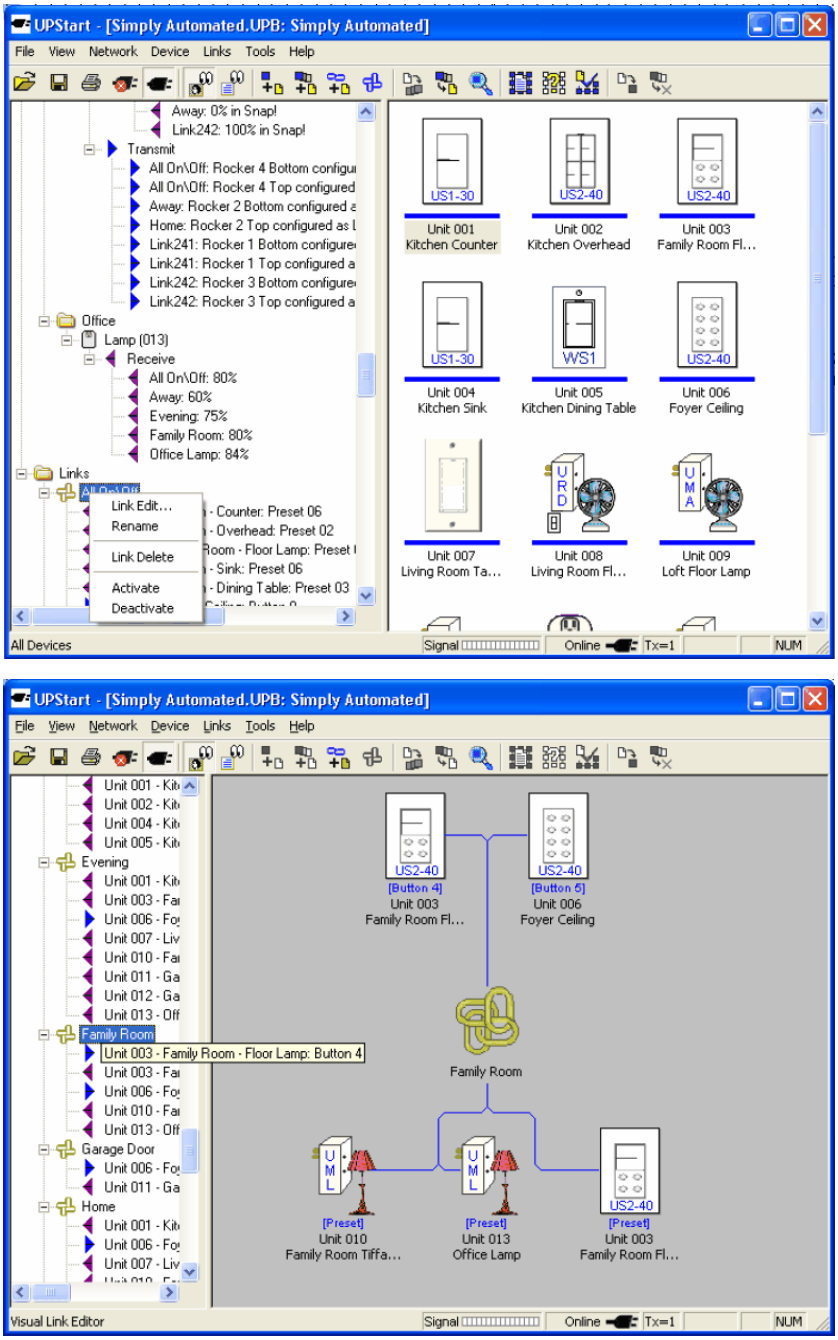


Figure 91 - UPStart Configuration Software

WebCTRL

www.automatedlogic.com

Automated Logic's WebCTRL is a multiplatform building automation server for BACnet network.

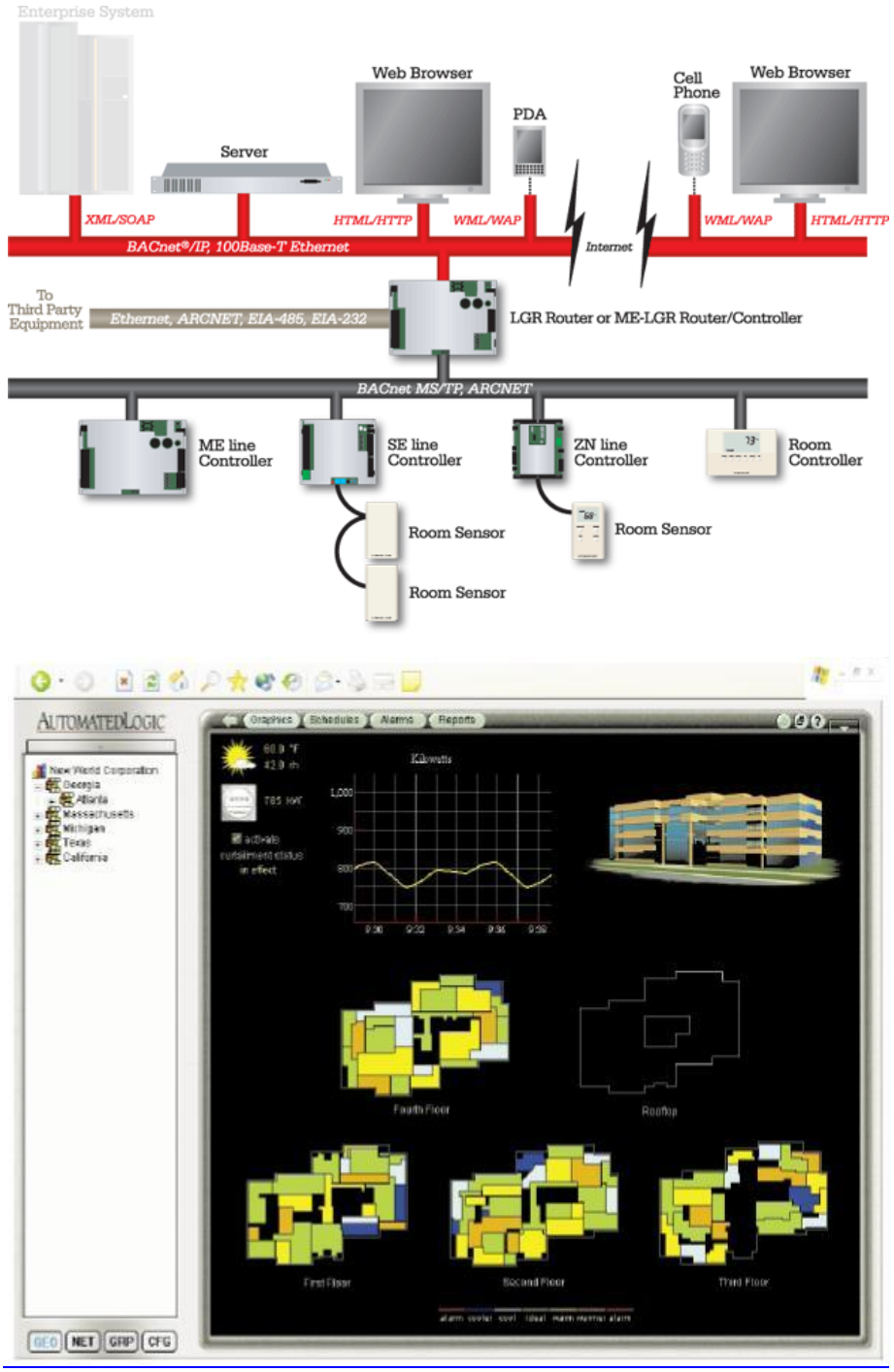


Figure 92 - WebCTRL

XTension

www.shed.com

XTension is a MAC application designed to bring X10 home automation to Macintosh users. In addition to common features, it allows AppleScript macros.

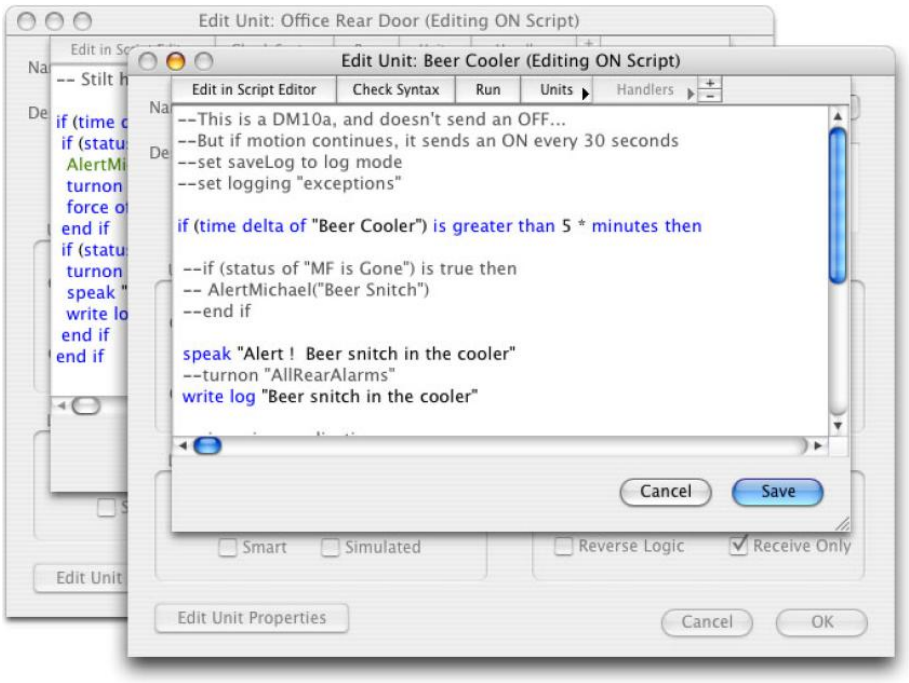


Figure 93 - XTension scripting

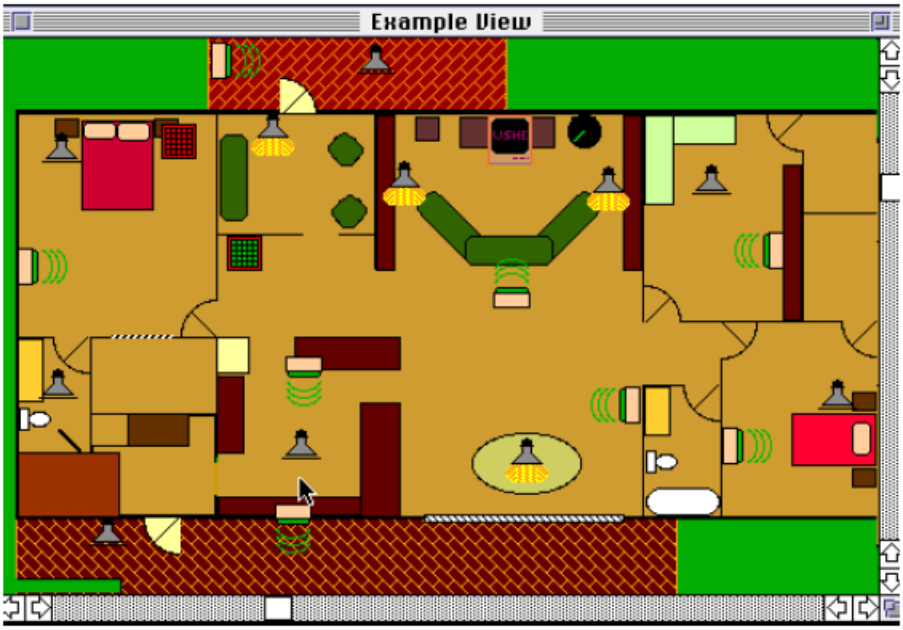


Figure 94 - XTension home view

Appendix B – Three layer control system idea introduction material

The aim of the following text is to introduce ideas proposed in this thesis to building automation providers. I use direct contact with control system developers from all around the world to get feedback on my work, contribute to the international research and promote Czech Technical University.

Jan Dolezal, dolezal.jan@gmail.com

Czech Technical University in Prague, 12/2007

Design of Smart home Multi-layer Control system

Motivation

The progress technology has made over past years is amazing. Computerization completely changed the work of billions of people in less than 20 years. But it did not affect their living as much. Why are we still so far away from the vision of ordinary households being as smart as the current technology and imagination allows?

I believe it is the lack of good software and solutions optimized for home use. Many of the current smart home installations have roots in the industry and some people believe that what can successfully control a whole factory should control a household with ease. The requirements are actually completely different. In industry, we usually need extremely high performance with high speed and stability. On the other hand, the simplicity of user interface and cost are amongst the main drivers for smart homes. The market of a billion households is undoubtedly big enough to get a specialized solution, tailored to its specific needs. Still most of the current specialized smart home software is either not intuitive for the end user or too limiting to advanced functions.

In order to launch mass installations and reach the economics of scale, we should consider different approach than has been used for individual installations in luxurious villas. The system must be cheap, reliable and easy to run and maintain by the user himself. Not before then will home automation become a standard, rather than a luxury or half functional do-it-yourself experiment implemented by techies. In my thesis I suggest such software.

From users to layers

A traditional control system consists of two layers: a rather sophisticated bottom layer with control algorithms and a simple top layer visualization with end-user interface. Whilst suitable for industry, it does not fit the users of home automation as can be seen on a simple example. Let's suppose an ordinary user bought a new wireless switch for the bedroom and wants to configure it so that it would turn off other lights in the apartment, set temperature ideal for sleeping and slowly dim the bedroom light within next 15 minutes. The top layer visualization is too simple to accommodate such a request, so the user would have to go into the bottom layer, which is often too complicated and not safe to modify by someone without good knowledge of scripting and all dependencies of the control system. So an average user cannot, but opt out to call an expensive integrator to set it up. This works in a few individual installations, but what if a million users decide to add a switch?

Simplifying the bottom layer for end user brings many compromises and limitations in functionality in many current specialized home automation installations. On the other hand, a complex top layer SCADA scares off inexperienced users. As a solution, I suggest to add one more layer to the control system, so that bottom layer stays a complex control system for critical algorithms that only an expert can modify (such as HVAC), middle layer allows common maintenance to advanced end-user and the top

layer serves as a very easy to use interface for computer-unskilled users such as children and elderly people.

Middle Layer

The proposed middle layer is an application running parallel to the bottom layer, which can be accessed remotely from any computer. It is designed for the growing number of advanced users, who are able to use a computer on the level of creating a PowerPoint presentation, but have no knowledge of control systems. Middle layer is connected to selected data points from the bottom layer and those can be assigned contextual information and used in a simplified logic to create rules. This logic implements basic lighting and switch logic in a visual way, which is more intuitive than common textual IF..THEN programming. Concerning HVAC, we use the middle layer solely for setting the target temperature according to button press or time schedules, but all the algorithms connected with achieving this goal are kept in the bottom layer.

For example, a grandma living alone could call her grandson that she feels cold in the kitchen when she wakes up at 5 am and has to increase the temperature manually every time. He connects to her middle layer application remotely from a PC at his dorm to see that the optimized heating was set to safe energy by decreasing the heating between 11 pm and 6 am. With a couple clicks he can increase the early morning temperature for her. Grandma actually likes her smart home, because she has no complicated controls or computer screens there and everything works even easier than before the smart installation. Grandchildren can easily help her to do any change, but they cannot make any big damage to the system, because they cannot access the bottom-layer function they don't understand. Furthermore, middle layer is technology-independent, so that it can integrate multiple bottom layer systems under unified interface (*for example simple X10 / Insteon network with HVAC control*).

Cooperation

Ideas presented above are further discussed in my thesis, which provides a detailed definition of a three-layer control system optimized for home automation, including sample middle layer interface, tips for hardware realization and other ideas about functions of smart homes. If you find this topic interesting, I will be glad if you check my thesis and let me know any feedback. I am open to cooperation.

About the author

Jan Dolezal is a master student of Control Systems at Faculty of Electrical Engineering at Czech Technical University in Prague and focused on home automation research under professor Burget. He is the author of the award-winning Windows Vista Media Center interface for home automation control and led a team developing a bottom-layer control system. Interested in connecting business with technology, Jan also have experience from studies at Union College in the USA and Zhejiang University and Shenzhen University in China.

Index