Control Architecture for Intelligent Houses

Steffen Larsson, Henrik Hansen, Lars Bo Theilgaard Madsen, Brian Ank Mertz and Christian Fink Petersen
Aalborg University, Department of Control Engineering, Email: {slarsson, henrikh, lbt04, bmertz and chr04}@es.aau.dk
Supervisor: Per Printz Madsen, Aalborg University, Department of Control Engineering, Email: ppm@es.aau.dk

***Abstract*— Home automation systems has increasing popularity, but the solutions are limited to certain areas like heat or ventilation and from a single manufacturer. These systems are made as individual systems and as such not made to share information about their own functioning. To make a complete system communication between these subsystems has to be enabled. Information gathered in one subsystem can be of value in another subsystem, thus sharing information can optimize the individual system, as well as increase the functionality of the entire system. In order to define the flow of information between the subsystems, a control architecture is established. A communication method is defined to standardise this information flow, and the functionality of the software facilitates this communication. This suggestion makes it possible to combine the individual subsystems into an integrated home automation system.**

I. Introduction

The tendency in modern household is automation of an increasing number of functions. This automation is administrated by different subsystems and these subsystems can be fabricated by different manufacturers. A subsystem is a fully working system, but limited to controlling a certain set of functions, it could be controlling the temperature in a certain room or the ventilation in an entire house. A subsystem often consists of a controller and a set of nodes communicating with that controller. This type of controller is referred to as a subcontroller. Nodes are often actuators or sensors in some form. The communication between the nodes and the subcontroller can be based on different communication platforms i.e. Bluetooth or Z-Wave. Even if subsystems are based on the same communication platform, some of them are still not able to communicate due to the variations in their data handling. Translating these variations in the communication into a standardised communication method can provide a solution to base an integrated home automation system on.

A solution should be independent of the underlying communication platform but it is still relevant to base the solution on an existing technology to absorb qualities from the communication structure. If the technology the solution is based on is attractive to the manufacturers, it is more likely to be used. Several different communication platforms have been compared on a variety of characteristics to determine which technology provides the best basis. The technologies compared are Bluetooth, Wavenis used in the LK Wireless home automation system, ZigBee and Z-Wave based on [1], [2] and [3], the comparison table can be seen on figure 1.

Z-Wave provides many of the desirable properties and is chosen as the reference technology. Its only lacking is its data transfer rate compared to other competitors, but as it is already used effectively in subsystems, the transfer rate is considered sufficient.

Platform	Flexibility	Reliability	Price	Power consumption	Transmission
Bluetooth	✓ ✓	✓ ✓ ✓	✓	✓	✓ ✓ ✓
Wavenis	✓	✓ ✓ ✓	✓	✓ ✓ ✓	✓
ZigBee	✓ ✓	✓ ✓	✓ ✓	✓ ✓	✓ ✓ ✓
Z-Wave	✓ ✓	✓ ✓ ✓	✓ ✓ ✓	✓ ✓ ✓	✓

Fig. 1. Comparing different home automation manufactures and their communication platforms.

A command structure, either distributed or hierarchical, needs to be defined in order to describe the flow of information and to define how much control needs to be in the subcontrollers and how much is needed elsewhere. This has a direct impact on how the communication gets handled by the system. To handle the translation from the different communication methods, both for different platforms and for similar platforms, a middleware software layer needs to be added to the system. These properties should enable communication between the subsystems as well as providing an interface toward the application layer of the entire system. This leads to the hypothesis:

Implementing a control architecture using a software layer can integrate individual subsystems into a functioning home automation system.

This should result in a system as represented on figure 2 with services provided for the application layer by the middleware and working on different communication platforms.

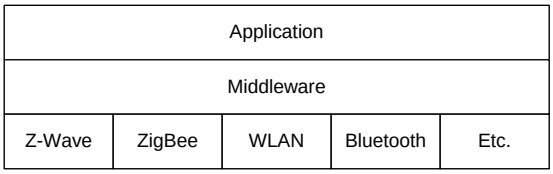

Fig. 2. The middleware is implemented between an application and different types of communication protocols.

The middleware provides an interface towards the application layer and towards the subsystems. A communication frame needs to be designed as well, in order to create a specific standard the application layer can use to access the services provided by the subsystems.

An example of the workings of a structure like this could

be a system with a heat pump and a solar panel on two different subsystems. The systems can provide set points, heat production and measurements available for the application layer, which can then optimize the cooperate functioning of the subsystems.

II. Methods

A. Control Architecture

Integrating individual working subsystems into a single network, requires specification of the control architecture in the network. One of the primary objectives of the control architecture is to provide a transition from the current partly- or non-automated systems to a fully integrated home automation system. With this in mind, a structure has to be chosen which, while still providing the full utility of a complete system, can offer information sharing on whichever level the subsystems are ready to handle it. Possible solutions could be either distributed and hierarchical structures.

The advantage of the hierarchical architecture is the minimal influence needed on the already existing and working systems. By adding an additional controller which the subsystems are assigned to, the subsystems can continue their existing functionality, while providing extra system wide services at little effort. This primary controller also provides a single entry point for a user interface, as well as giving easy access to the data storage, frameworks and applications for upgrades and additions.

A distributed architecture implements the possibility of sharing control between the subsystems. This also implies access to all services on all subsystems. An approach like this requires more communication between subsystems to stay updated. The more communication, the higher the risk of collisions and wait time involved in retransmitting, thus limiting throughput for the system. The system wide applications and data storage would also have to be distributed, complicating the decision process. Upgrading the system based on this architecture will gradually increase the communication needed between the subcontrollers. Adding an application layer to a distributed architecture, would also require either a complete application on all subsystems or partial applications with a lot of communication. If the system is expanded onto more than one platform, a distributed system would require a hardware bridge on every subcontroller in order to be able to communicate with all other subcontrollers. It induces a lot of hardware bridges, if an increasing number of subsystems consists of different types of hardware platforms. An illustration of such system can be seen on figure 3.

One of the main advantage of the distributed architecture is the immediate access to information. In spite of this, the hierarchical architecture is chosen as the preferred structure, because of the advantages described above. This results in the need of a primary controller added to every system with 2 or more subsystems. It also requires additional software, referred to as middleware, added to the subcontrollers to facilitate the communication upwards in the hierarchy. This is illustrated on figure 4.

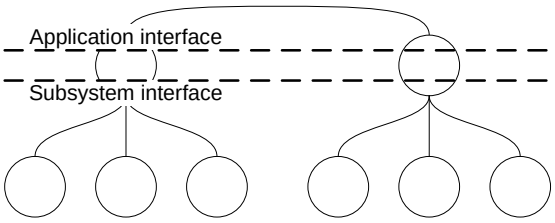

Fig. 3. Distributed control architecture.

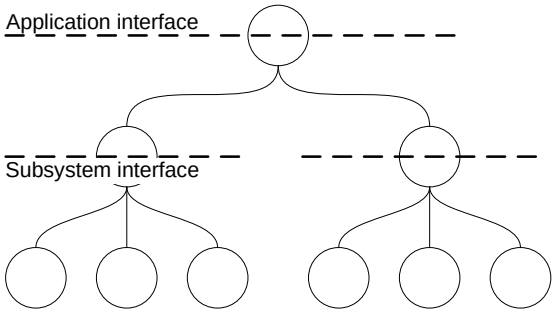

Fig. 4. Hierarchical control architecture

B. Middleware

Middleware is an application programming interface (API) that makes subsystem services available to the application, and makes it possible to exchange data between applications. Just like e.g. the internet, services are provided to different browsers across the globe, and along with the TCP/IP protocol ensures that the underlying platforms remains invisible for the end user, regardless of platform type.

Middleware for home automation is based on the same idea, enabling the control architecture and providing services for the application, regardless of the underlying platform. As the control architecture assumes a hierarchical structure, with the interface towards the subsystem on one level and the interface towards the application on another level, the middleware is split across the controllers. This way the middleware have to handle internal communication.

The subsystem is connected to the primary controller using the platform of the subsystem, in this case the Z-Wave platform. When the controller in the subsystem receives information from its underlying nodes in that subsystem, which the subsystem application deems necessary for the system application, the information is translated in the middleware to a standardised format. The information is then encapsulated in the Z-Wave frame and transmitted to the primary controller. The primary controller processes the information in the Z-Wave protocol layers and present the information for the middleware as it was sent from the subsystem. The middleware

then presents the information to the application layer, then it can be accessed by programs running on the system. This is illustrated in figure 5 using the arrows marked with 1.

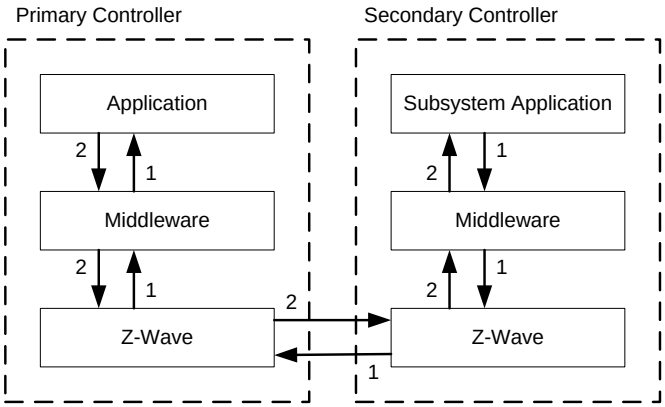

Fig. 5. The functionality of the middleware

When an application running on the system needs to access a service in a subsystem, the request is made to the middleware in the primary controller. The middleware in the primary controller keeps track of which subsystem and which parameters are associated with a certain service, and what communication platform this subsystem uses. When this is known, the middleware can create the standardised frame, in this case encapsulate it in the Z-Wave frame, and send it to the selected subcontroller. In the subcontroller the frame is decapsulated and the standardised frame is translated by the middleware in that subcontroller into the communication type used in that subsystem. The subsystem then handles the information as given by the subsystem application and performs the requested service. Figure 5 illustrates this with the arrows marked with 2. These two transitions can be achieved using any arbitrary communication platform, as long as the middleware in the current subsystem is capable of translating the communication in that subsystem into a standardised format and vice versa.

C. The Middleware Frame

The proposed middleware incorporates a frame which can take three different forms, illustrated in figure 6.

The fields included in the frames are based on having a non-complex communication frame that offerers services for the application above. It is dynamic in its length and reliable in delivering the frame, which fulfills the request for flexibility described in the introduction.

Type Frame	Destination address	Sender address	Frame type	Length	Data fields size (X) + 1 byte		Checksum
Data	Unit ID	Unit ID	1	X*1 byte + 1 byte	Type	1 byte	Yes
Parameter	Unit ID	Unit ID	2	2	Type	1 byte	Yes
Acknowledge	Unit ID	Unit ID	3	None	None	None	None

Fig. 6. Middleware communication frames.

The destination and sender address fields are intended to uphold unique identification of the components communicating directly with the primary controller. This approach is used in order to distinguish addresses from different technologies. Among other things the frame type is intended to give the opportunity to send and receive either parameters or data. Parameter frames are sent from the primary controller in order to make use of services offered by the subsystems. Data is used when information is sent from the subsystems to the primary controller. Each frame must be acknowledged before the next frame is sent in order to ensure, that the destination has received the frame. If the acknowledgment does not arrive, the sender experiences a time out, and resend the frame. The acknowledge frame does not have to include length, data and checksum fields, which entails a shorter frame and less communication. Like the acknowledge frame, the parameter frames always have the same length, while the data frames vary in length. The length of data frames depends on the size of the data being sent. The checksum field indicates if the entire frame is received without error. Since the communication protocol includes error detection too, the checksum field in the middleware frame is included to add security to communication platforms without this security.

In order to transport the middleware frame between the middleware in the subcontroller and the primary controller, the frame needs to be encapsulated in the transport protocol frame on the given communication platform. This way the underlying network will provide the transportation. Encapsulation is used when upper layer data is passed down a protocol stack. The concept of encapsulation is shown on figure 7 using the Z-Wave platform.

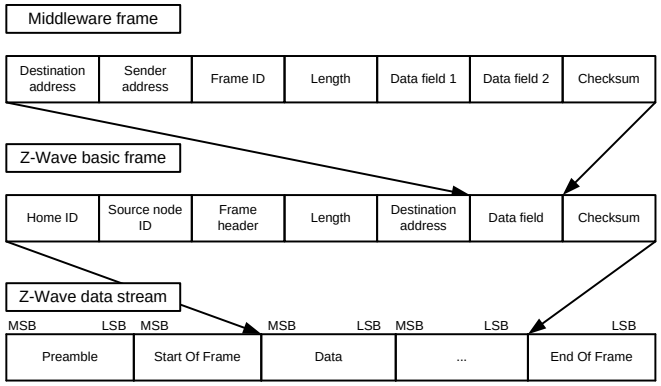

Fig. 7. Encapsulating the middleware frame and transmitting it via the Z-Wave platform [4].

D. Simulation

A simulation of part of the Z-Wave platform and middleware has been implemented to get a proof of concept, mapping the middleware on to the physical network and still achieve the intended functionality. The simulation is based on the assumption that the simulated subsystems are designed by

different manufacturers and not able to share information. To link these subsystems the middleware is implemented as a solution to enable the communication.

The simulation has been done without taking into consideration, that the subsystems has limitations due to noise, time delay, package loss etc. Furthermore it is assumed, that the middleware frame fits into the frame that encapsulates it. The focus is only on how packages are distributed in the network.

The simulation shows how the middleware distributes information and data by the use of a hierarchical communication structure. The distribution of the data is done by encapsulation a middleware frame containing data from one subsystem into a Z-Wave frame and transport it to the primary controller. The middleware in the primary controller decapsulates the frame and an application processes the data. Decisions made in the application contains instructions which are sent using the middleware frame. The frame is then encapsulated again into a Z-Wave frame and transported to the relevant subsystem. In this subsystem the same procedure concerning decapsulation takes place.

E. Performance

As mentioned in the introduction the Z-Wave data transfer rate is limited. Therefore an investigation is made in order to show how encapsulation of a middleware frame affects performance in the network.

Figure 8 shows the Z-Wave frame with and without middleware frame, consisting of header, data and checksum. Encapsulation of the middleware frame entails an increasing header field of 4 bytes. The maximum length of data specified by Zensys is 47 bytes [5]. Including the middleware frame the length of data can be at 43 bytes at maximum.

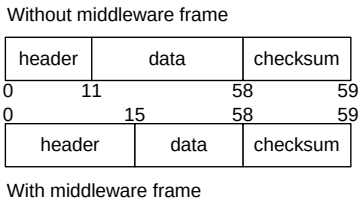

Fig. 8. With and without encapsulation of the middleware frame.

The goodput is defined as the amount of data pr. time unit from the application level that can be transfered from sender to destination. Goodput is used as an expression of the performance, with and without the middleware frame, in the network and calculated from the following equation.

$$B = \frac{D}{T} \tag{1}$$

Where:

B = Goodput
D = Usefull data
T = Mean transfer time

To investigate how data length effects the goodput, calculations for values between 4 and 47 bytes are made for the Z-Wave frame. The last 4 calculations are left out on the graph. Same principle is done when encapsulating the middleware frame, but calculated for values between 4 and 43 bytes. The graph on figure 9 shows the goodput as a function of data length.

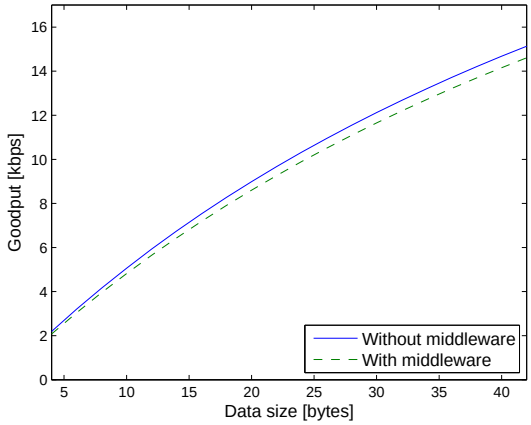

Fig. 9. Goodput as a function of the data length.

The difference in goodput for the two plots increases as a consequence of greater data length. However, the differences are between 5,5 % and 4 % for small and great data lengths respectivly.

III. Conclusion

The middleware and control architecture can be applied to numerous systems, however it is a huge area with complex requirements and only part of it has been covered. The hierarchical control architecture provides the most efficient structure for implementing a home automation system, due to minimal impact on existing solutions, while providing the benefit of top-down control and one-point access. Each individual subsystem has their own controller, which manages the applications running in that subsystem, while the primary controller handles system wide applications. This way, existing systems can be integrated in to a complete system, providing the possibility of upgrading to this solution at low cost.

The middleware is responsible for translating the subsystem communication data into the middleware frame format. The middleware in the secondary controllers is designed to access the application in the subsystem, communicating messages to and from the primary controller. On the primary controller the middleware is designed to provide an interface to the system wide applications. The middleware frame was purposely designed to be as flexible and non-complex as possible in order to fit as many manufacturers as possible. Based on the analysis and design it is believed that this has been achieved. If the standardised communication method is followed a link between the controllers is established. Then the integration of

the subsystems into a complete home automation system is possible.

A simulation of the physical communication between two subsystem has been performed. The simulation showed that it was possible to get information from two subsystems, through the middleware, utilizing the middleware frame. The simulations shows proof of concept for the middleware communication.

Using the middleware frame in conjunction with the Z-Wave platform has a limited effect on the goodput of the transmission. As Z-Wave is the technology with the lowest bandwidth, the implementation of the middleware frame is assumed possible on all types of examined subsystems, without particularly affecting performance.

Due to the nature and scale of the system, it is not known if the middleware created is able to support all the different manufacturers, and it cannot be determined if the structure chosen will work with all systems, or if the frame supports every data type. The potential of a system like this is the opportunity to combine specifically tailored systems, already on the market from different manufacturers, covering different areas in home automation, heat, lighting, security, ventilation, etc. This will provide subsystems from manufacturers specialized in their area of expertise.

IV. Further Development

Further developments involves extending the middleware to incorporate interfaces to different manufacturers, as it requires acceptance from a wide variety for it to be fully integrated as an accepted standard.

A complex network of subsystems consisting of different platforms could experience heavy load on certain channels if there is too much traffic. As some of the communication platforms have limited bandwidth, a solution to an overloaded channel could be adding additional hardware to a subsystem enabling communication using e.g. Bluetooth even if that system is already using Z-Wave. The middleware would have to be extended to also include choice of channel depending on the current traffic on that channel, thereby increasing bandwidth and avoiding critical situations.

Acknowledgment

The authors would like to thank for help and advise:

Associate Professor, PhD, M.Sc.E.E, Per Printz Madsen

Roozbeh Izadi-Zamanabadi, R&D Control Specialist, Danfoss A/S

Søren Hansen, Manager Communication Technology, Danfoss A/S

Associate Professor Ole Borch

References

[1] Mark Walters, http://www.z-wavealliance.org/modules/start/, Webpage date: 16 oct 2007.

[2] Penton Media Inc, http://rfdesign.com/next generation wireless/news/zwave vs zigbee/?cid=zigbee, Webpage, date: 22 oct 2007.

[3] Morten K. Thomsen, Kamp om standarder til automatiske hjem, http://ing.dk/artikel/76981, Webpage, date: 23 nov 2007.

[4] Zensys, Z-Wave Protocol Overview, 2007, Confidential literature

[5] Zensys, Z-Wave Transmission Time Calculation, 2007, Confidential literature

Control Architecture for Intelligent Houses

Worksheets

Christian Fink Petersen

Henrik Hansen

Brian Ank Mertz

Steffen Larsson

Lars Bo Theilgaard Madsen

Group 726
E7-project 2007
Department of Control Engineering
Aalborg University

Contents

Chapter 1

Scope of the Project

This project is about designing a network communication infrastructure build on top of the Z-wave platform targeted at an intelligent house. That involves investigating different types of network topologies to determine the best way to organize the multitude of wireless units meant for all the electronic equipment in a house. The Z-wave units are already capable of receiving and transmitting/retransmitting information which leaves the layers above the network protocol level to be designed.

As the wireless modules are powered by battery, there will be some central units connected to the grid, which will be retransmitting information when needed. That way the short range units will only transmit their own signals, while the designated modules will transmit all the received signals to ensure that all the units have access to needed information even if the transmitting unit is out of range. Eventually all the modules in a system should be configurable from a central computer/webpage using a graphical user interface.

To make this work across different subsystems from different manufactorers, a software layer needs to be built on top of the protocol layer already specified by Zensys. This layer should provide information and communication on a logical level to ensure compatability.

Chapter 2

Background

In modern houses more and more functionalities are being automated. Is is everything form ventilation and lighting to heat control. There are a multitude of solutions available from many different manufacturers covering different areas in home automation. Many of the existing systems are wired but it is desired to make the systems wireless and therefore less expensive and extensive to implement cablewise.

In that matter, an optimal solution will be some cheap wireless modules that can easily can be implemented in existing hardware. As an example, Z-wave modules from Zensys can be used.

A number of companies has already designed home automation systems based on the Z-Wave communication platform, such as Innovus [1]. But it cannot be taken for granted that systems from different companies can communicate with each other even if they are using the same platform.

[1] www.innovus.dk

Chapter 3

Hypothesis

Implementing a control architecture using a software layer can integrate individual subsystems into a functioning home automation network.

This project is based on the project proposal by Danfoss A/S on control architecture in a wireless home automation system. The proposal is based on a setup consisting of a heat pump and a few heaters and the general idea is that if enough information is shared between the elements in the system, it is possible to optimize the system and thus the energy consumption.

The control architecture is the logical communication structure, in this case built on top of the Z-Wave protocol layer made by Zensys. The Z-Wave physical structure and lower layer communication protocols are well-defined and has been working for a few years already. The control architecture is relevant when isolated systems made by different manufacturers is to be combined into a working home automation system with some sort of central control and sharable information. At the moment these systems, even though based on the same physical structure, cannot share information. Hence the goal is to develop a standardised software layer which can facilitate the communication across subsystems. This type of software is also called middleware, a layer between the databases and the client in a multitier architecture. By definition, middleware hides the distribution and differences in the underlying systems by providing a common platform [Tanenbaum and van Steen, 2002, p. 36].

Chapter 4

Problem Analysis

When having an installed home automation it is possible to set up some scenarios to control certain function at certain times. Such scenarios is describe in appendix A. In the sense that components from different manufacturers have to communication with each other, it is desired to have a logic communication architecture which makes it possible to add/remove components to an existing system.

As an example a heat system contains of a heat pump and two radiators from a certain producer. Later on a solar panel is being added to the system. Such a system is illustrated on figure 4.1.

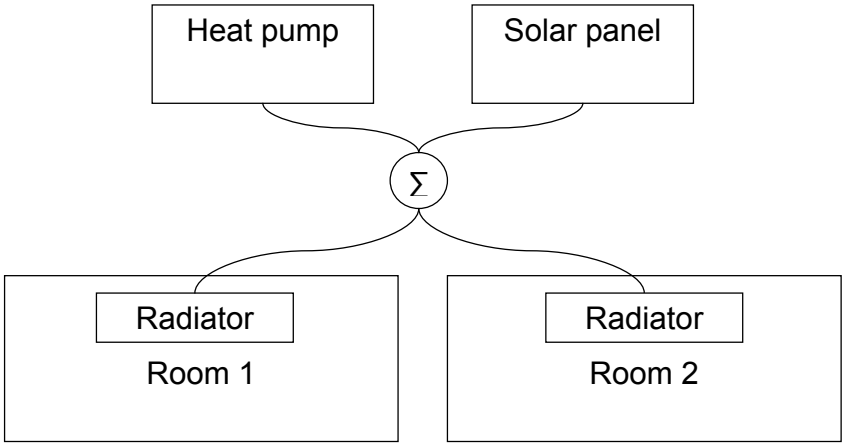

Figure 4.1: *Scenario of the system. The system consists of two heat sources and two radiators*

The heat pump and the solar panel are now more or less running in parallel to supply heat to the system. Due to the fact that the sun do not always shine, the system has to have a controller which decides how much from each source the heat should come from. To makes the action possible, the levels on which the components are communication has to specified.

Further more, the room whose temperature set point is the highest, decides the water temperature in the system. The radiator in the other room is then adjusted by a valve.

The physical communication architecture has already been designed by the developer of the Z-wave modules. So from this point the logically communication architecture has to be designed. To do this some different approach will be analyzed.

4.1 Logic Communication Between Nodes

In a house control system the Z-wave nodes are sending each other some information or some commands. The Routing of the signals is already designed[1]. Figure 4.2 shows an example of a physical routing and the logic routing of a signal. Depending on the configuration of the system there might be a mains powered node on the way from A to B, which receives and transmits commands on the network. When speaking of logic communication command it is these that

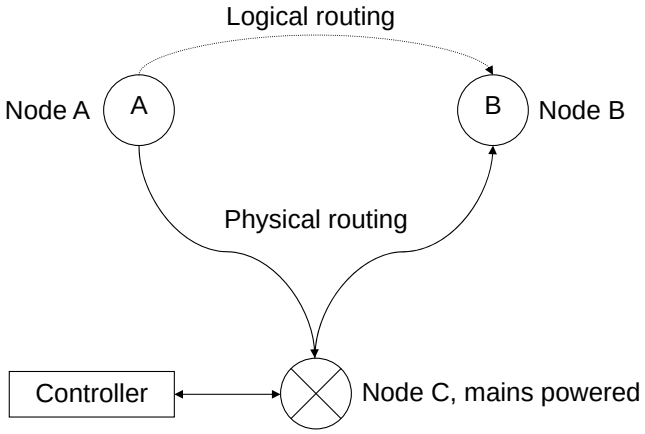

Figure 4.2: *Routing of a signal from A to B. The dashed line is the logic communication and the solid lines are the physical communication.*

are used in the applications. For example a temperature sensor can send some information to a valve directly. Another way of doing this is by sending information through a controller. In this case the node must transmit its own ID, the value and who the receiver of the information is for.

Properties of Using a Controller

The advantages of sending through a controller is that only one command has to be send to the controller and the algorithm in the controller makes the choice which node are going to receive the command.

Further more the controller makes it possible to divide nodes into two groups, sensors and actuators. In the example with the heat pump and the solar panel, they are considered as actuators. In this case it is possible to make some priorities and choices in the controller depending on the actual situation.

[1] The physical communication has been designed by Zen-sys.

Properties of Direct Communication

When using direct logically communication the sensors sends commands directly to actuators. The actual physical communication is passed through a controller. In this architecture the logically structure are said to be flat.

The benefit of this approach is that modules are equally likely. Modules from different producers are able to communicate directly with each other.

4.2 Control Architecture

The control architecture is the structure of the network. With the physical system as well as the lower level protocol layers and routing mechanisms already defined, the control architecture handles which nodes are allowed to share information with certain other nodes as well as providing a command structure. There are 3 different ways to set up the logical structure of the network. It can be hierarchical like the physical structure, nodes are assigned to a controller to create subsystems and those subsystems are then linked to a primary controller as in figure 4.3. The completely opposite way of it doing it is to allow communication on a logical level between all nodes across subsystems as on figure 4.5. The third way of setting it up is to allow communication between the controllers of the subsystems, a semihierarchical network as in figure 4.4.

There are several things to consider when choosing the structure of the network. Accessibility of data is a key point, as the entire purpose of providing the control architecture is to make sure data is distributed to the necessary nodes and subsystems to optimize performance of the system. This is generally a feature of a distributed network. Directly related to data accessibility is dataoverflow. With limited bandwidth and limited storage capacity on the battery driven nodes, there is a limit to how much data can be shared. Another key point is conflict of interest. Different subsystems can with the same data take different actions. If these actions then affect the same node or subsystem, a decision will not be taken if there is not a primary controller taking it.

If the system is to learn over time, it is necessary to have some form of tracking or logging feature which can, based on previous measurements, predict certain changes the system will be exposed to. This also has an impact on how the control architecture is implemented, as predictions affects the decision making and logging needs to be centralized in some form to avoid stacking huge amounts of data on all nodes.

4.2.1 Hierarchical Structure

Similar to how the lower level protocols and physical structure of the Z-Wave system is designed, the hierarchical control architecture has a primary controller and some nodes. For larger systems, the primary controller controls some secondary controllers which then supervise their respective nodes. This way the system contains a number of subsystems all connected to the primary controller. The communication flow is always up and down, never between nodes on the same level, which means that all data passes through the primary controller if it is moved from

one subsystem to another. Internally in the subsystems all communication passes through the secondary controller.

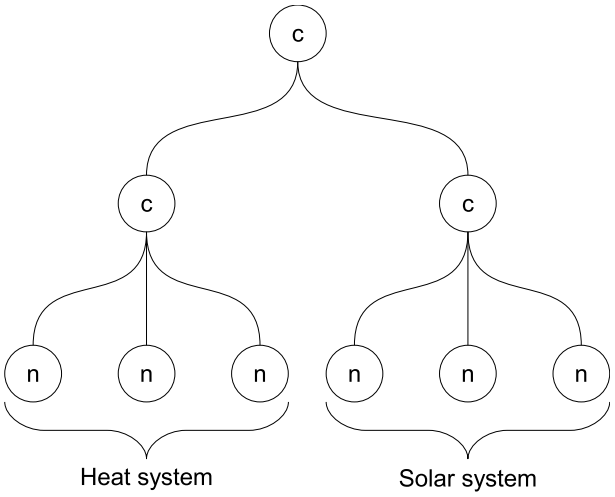

Figure 4.3: *The figure is illustrating the principal of the logical communication based on a hierarchical structure.*

The most striking advantage of having a hierarchical structure is the resemblance to the system already given by Zensys. Building on top of the Zensys platform however is not given if the architecture is to be used more widely than just the Z-Wave platform. A structure of this type never has any control conflicts. Every decision is made from the top, but this also means that the primary controller needs to have access to all the information in the system. This is a fairly easy task on the Z-Wave system as all communication is passed on to the controller from the nodes for further distributing. But this also means that every message generated by the system have to go through the controller. The Z-Wave structure limits this kind of communication as well as the battery driven nodes are not listening all the time and then cant communicate which each other. For a strict hierarchical structure of a certain size, the everlistening secondary controllers cannot communicate with each other as well, they have to pass the information through the primary controller.
If the hierarchical structure is chosen it has to be decided what level of control the controllers will get over their minions. There are several factors involved in this

4.2.2 Semihierarchical Structure

The objective of the semihierarchical structure is to simplify the routing between the subsystems by decreasing the number of hierarchical levels, which is done by removing the primary controller. Instead of having a primary controller, the secondary controllers is able to communicate directly. The semihierarchical structure is illustrated in figure 4.4. The communication flow in the subsystems is handled similar to the hierarchical structure.

The advantage of utilizing a semihierarchical structure is the secondary controllers are able to communicate without passing the messages to a primary controller first. This will lead to a more simplified routing compared to the hierarchical structure, but also introduce issues of how the communication flow is handled without a primary controller. If this type of structure is chosen, all secondary controllers has the same hierarchical level which can cause control conflicts. To

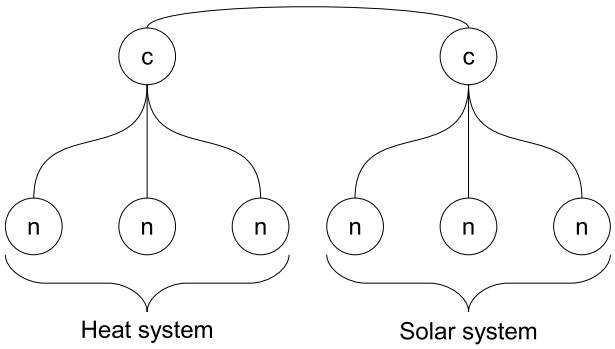

Figure 4.4: *An illustration of the logical communication based on a semihierarchical structure. The secondary controller of each subsystem is able to communicate directly.*

avoid conflicts it is necessary to determine an interface between the controllers and in which order each controller should take the status as primary controller. Because the primary status is taken in turn, each secondary controller has to contain information of which controller that has the primary status and the services other subsystems offers.

4.2.3 Distributed Structure

Figure 4.5 illustrates a distributed structure. The idea of the distributed structure is all nodes are treated equally, which means nodes from one subsystem are able to communicate directly with nodes from other subsystem. Because all nodes are equal the system do not contain any static controllers. Instead of having controllers the nodes will get the controller status in turn.

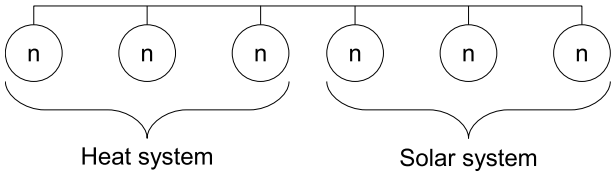

Figure 4.5: *Illustration of nodes implemented in a distributed structure. All information are available to every node in every subsystem.*

The benefit of using a distributed control structure is the availability of information due to the nodes can communicate directly. This means that a node can request information from other subsystems without requesting the secondary controller first. Another advantage is the transfer time of a message can be decreased compared to previously described structures. A distributed control structure is having some of the same design issues as the semihierarchical structure, because the structure do not have any higher level controllers. As mentioned earlier, an order of receiving the primary status is needed to avoid conflicts between nodes which has the same level. Due to each node needing to have knowledge of primary status and memory to store the provided services, the informations about status and services has to be redistributed to all nodes when changes are performed. Depending on the size of the system the sharing of information can entail a large amount of communication between the nodes, which may decrease the performance of the system and the battery life. The choice of how the communication flow is handled will also have impact on the scalability of the system. If the system is extended with extra nodes,

all other nodes has to be informed about the modifications due to the knowledge of services. It is also required that the transmitting order is updated.

4.3 Existing Solutions

In this section four technologies will be discussed in order to decide which one of them that will be used to fulfill the needs, discussed in the scenarios. The technologies are Bluetooth, LK Wireless, ZigBee and Zensys. Five criteria have been chosen to be relevant, and should give a comparison between the technologies. These are: flexibility, reliability, price, power consumption, bandwidth. The sending ranges are rarely the same, around 10-30 meters indoor, and not taken into consideration. It is important to notify that specially the first two criteria are "quality aspects" and are based on subjective evaluations of articles and web pages.

4.3.1 Bluetooth

Bluetooth has for a long time being considered to be the wireless standard for home automation. It is a quite flexible technology with a relative high reliability, but it is a much more expensive solution than the following technologies. It has a relatively high power consumption which might be caused by the high bandwidth at 723 kBit/s. Since wireless home automation require good battery lifetime this is perhaps why Bluetooth is only preferable between low number of devices e.g cell phones and computer.

4.3.2 LK Wireless

IHC Wireless form LK is the only producer of this closed standard for home automation. It is only possible to combine LK IHC Wireless with other LK IHC products. On the other hand it is a highly reliable solution. Since it has to be an integrated installation with e.g. switch button and receiver to lights, where prices starts around 50 US dollar, this solution can be a costly affair. Between every battery shift LK promise at least 5 year. The transmission rate is 2,4 kBit/s.

4.3.3 ZigBee

The last two producers that are discussed have much in common. Both ZigBee and Zensys have made an alliance in order to make an open standard for wireless communication. But Zigbee is not only concentrating on home automation but also in a industrial matter. The ZigBee alliance today counts more than 250 members.

Any device with a ZigBee chip inside can communicate with another chip, and vice versa. That is why flexibility is considered to be medium. Many companies in the alliance are trying to get their own imprint included on a wide solution. That is why ZigBee at the moment are not considered to be optimal 726 [2007]. Price for a ZigBee chip is found to be less than 5 US dollar, and expected to be reduced to the half when increasing production. ZigBee uses about 25 mA

when receiving/transmitting data, and can do this at up to 250 kBit/s Thomsen [2007]. This makes it possible to have video surveillance and other large data transmissions.

4.3.4 Zensys

The last producer is Zensys that has developed the Z-Wave network which is wireless communication between Z-Wave nodes. The Z-Wave alliance counts 125 companies, including Danfoss, and the main purpose is to develop a semi closed standard chip solution. It is specially designed for home automation and the idea is to use mesh technology so every node in the network can act as a router if disturbance should interfere the connection. This will prevent so-called dead zones.

The flexibility is considered to be medium since any devices can communicate as long as they all have Z-Wave chips inside Walters [2007]. Z-Wave is a very reliable technology 726 [2007]. The vision for the Z-Wave alliance is that an chip implementation in any device should be around 3-4 US dollar. As described in the further section, the architecture of the system means that it has a little lower consumption than ZigBee, at 23 mA when sending. The bandwidth is relative small, only at 40 kBit/s, which reduces the possibility of using it to e.g. web cams etc Inc [2007].

The four discussed types are evaluated from a scale 1-3 on figure 4.6.

Platform	Flexibility	Reliability	Price	Power consumption	Transmission
Bluetooth	✓ ✓	✓ ✓ ✓	✓	✓	✓ ✓ ✓
LK Wireless	✓	✓ ✓ ✓	✓	✓ ✓ ✓	✓
Zigbee	✓ ✓	✓ ✓	✓ ✓	✓ ✓	✓ ✓ ✓
Zensys	✓ ✓	✓ ✓ ✓	✓ ✓ ✓	✓ ✓ ✓	✓

Figure 4.6: *Comparison between different producers*

The Z-Wave is chosen in order to fulfill the specifications in the scenarios, and further details are described in the next section.

4.4 Choice of Communication Platform

Generally the Z-Wave network consists of two different types of nodes which are defined as controllers and slaves Zensys [2007*a*]. These two types can be in different variations but the main functionality for the controller units are to calculate routes for transmissions in the network. The slave units are basically just input and output units.

4.4.1 Controller Types

There are 4 types of controllers: Portable controller, Installation controller, Static controller and Bridge controller. Furthermore these can be either primary or secondary. Every Z-Wave network must have one primary controller, which are the first node added to the network. Following added are secondary controllers. It is possible to allow a secondary controller to be primary, but there can be only one at a time. The former primary will therefore be degraded to be secondary.

The portable- and Installation controller are much the same type. The Installation controller is basically just an extension of the Portable, which can be used in more advanced features. Both have the capability of finding their own location in the network and are typically battery powered. These features makes it obvious to use these controllers in remote devices in e.g. light, garage, window blind etc.

Regarding to its name, the Static controller must be in a fixed position and must not be moved physically. Furthermore it must be mains powered because it has to be in listening mode all the time. Thereby the static controllers becomes junctions in the Z-Wave network and can send/receive without battery considerations. When new nodes are added to the network, or some nodes unfortunately are lost, the user has to update the primary controller manually. This can be overcomed by adding a Static Update Controller (SUC) which automatically will map these changes. When other controllers in the network request for information, the SUC will release an update. In case the primary controller is lost or not working, the SUC are capable of assigning the primary role to another controller.

An extension of the SUC are the SUC Id Server (SIS). The SIS will then become the primary controller of the network, which also means that there can only be one SIS in each network. The SIS allows all other controllers to include/exclude nodes and therefore the Node ID's in the network must be maintained by the SIS.

The Bridge controller has the same features as the Static controller, but also the possibility to communicate with nodes through other types of network e.g. TCP/IP.

The Z-wave protocol 4.6 supports a network consisting of a maximum of 232 nodes. Furthermore it is possible to extend the number of networks with a backbone switch that is Z-Wave protocol compatible.

4.4.2 Slave Types

The Z-Wave network has also 3 different type of slave units: Slave, Routing slave and Enhanced slave. They can be either be driven by battery or A/C. Battery driven slaves are suppose to sleep as much as possibly, and only wake up from time to time, to decide whether it should react or go back to sleep mode. This is a consequence of the wireless technology: the battery driven devices needs to use as little energy as possible. If the slaves are powered by A/C it can be used as a repeater in the network.

As mentioned above the slave units are typically used in devices that only require inputs and output, for instance a socket. Slaves can also have a routing functionality. These can be either Routing slaves or Enhanced routing slaves, and their functionality is based on reaction on events.

The controllers have assigned them predefined routes which makes it possible for a slave to communicate with other nodes, when an event happens. A doorbell system is an example of how routing slaves, inside the switch/button, can activate the bell. The two routing slave type can be either A/C powered or battery. The first type of supply extends the slave role from being only listening, to be a repeater. The main difference between Routing slaves and Enhanced routing slaves is that last mentioned have software support for using External EEPROM and an RTC.

4.5 Z-Wave Software Architecture

The Z-Wave software is split into two groups, the Z-wave basis software and the application software and can be seen in figure 4.7 Zensys [2007*b*]. The startup code, low-level polling function, main poll loop, Z-Wave protocol layers, memory and timer service functions are running in the basis software. The application software includes the hardware and software initialization, state machine, completed callback functions and received command handler. The different layers and functions in the Z-wave modules will be described in the following.

4.5.1 Z-Wave System Startup Code

The startup code is located in the main loop and only active when the Z-wave modules are tuned on. The startup software initializes the Z-Wave hardware and the application hardware by calling the **ApplicationInitHW** file. The software initialization is done by execution the **ApplicationInitSW** file.

4.5.2 Z-Wave Main Loop

In the main loop the Z-wave protocol functions, the **ApplicationPoll**, the **ApplicationCommandHandler** and the RTC timer function running in a round robin order. All the functions must return to the caller as fast as possible. This means no busy loops are allowed, and it is possible to receive Z-Wave data, transfer data via the UART and check user-activated buttons at the same time.

4.5.3 Z-Wave Protocol Layers

A thorougher exposition can be seen in 4.6

4.5.4 Z-wave Application Layer

The application process software to the Z-Wave modules, is located in the hardware and the software initialization functions **ApplicationInitHW**, **ApplicationInitSW**, in the application state machine **ApplicationPoll,** and the command complete callback functions, and a receive command handler function **ApplicationCommandHandler**. The application layer provides

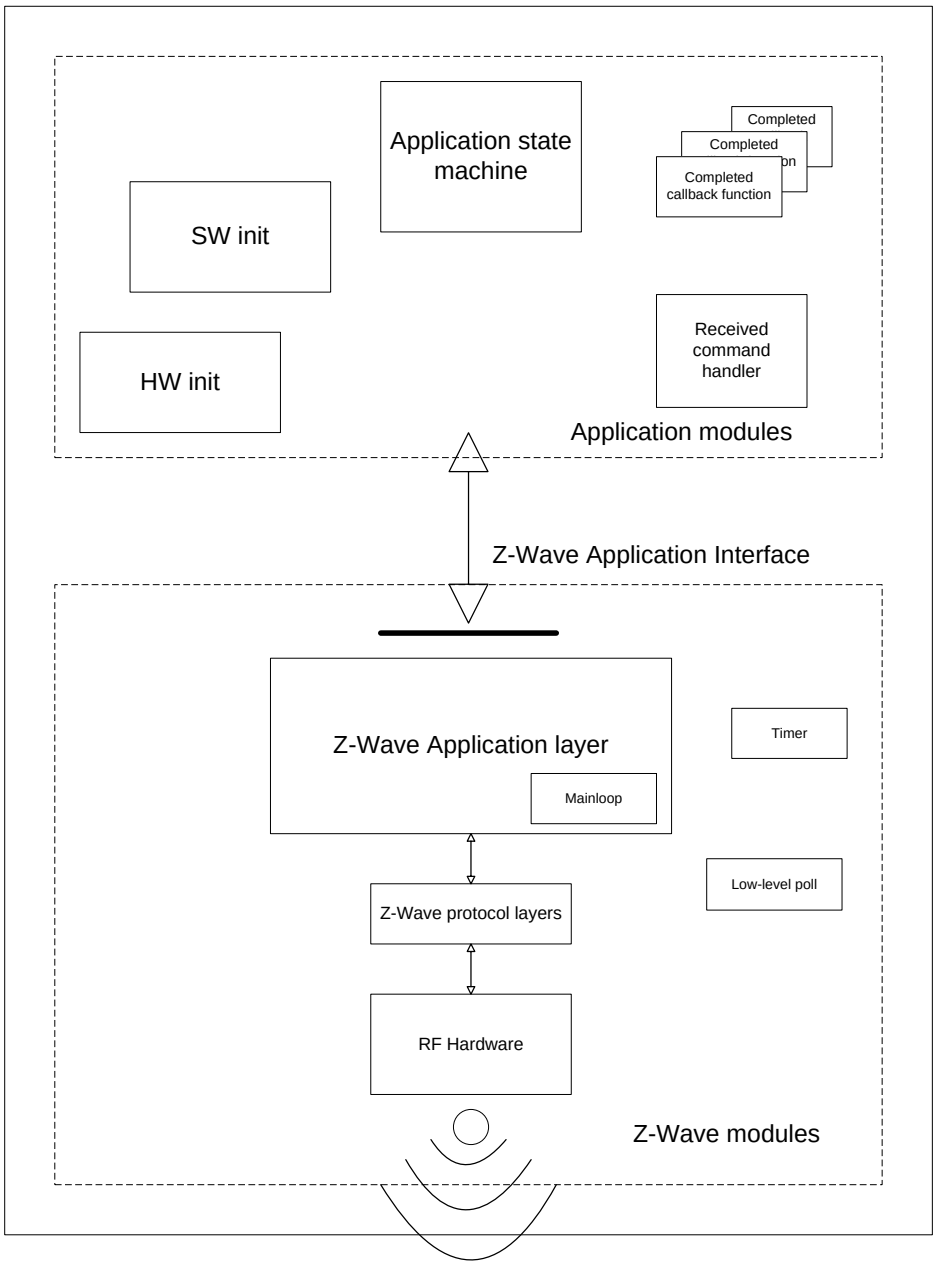

Figure 4.7: *The Z-wave Software architecture*

the ability to use the framework defined by Device and Command Classes. The commands in the Device and Command Classes provides the ability to to set parameters or request node parameters. The Device and Command Classes is created in the **ZW classcmd.h**.

4.5.5 Z-Wave Software Timers

The software timers can handle more tasks simultaneous, some of the timers in the software is reserved for some protocols timeouts. If a function calls the **TimerStart** API, witch saves the

function address and sets the timeout value and returns the timer-handle. If the function needs to cancel the time out, can it call the timer-handle.

4.5.6 Z-Wave Hardware Timers

The 3 generation of Z-Wave modules have different timers/counters. And different timers a reserved for different thing in the protocol, according to the Z-Wave modules model. The different hardware timers and the different models kan be see at 4.8

	Zw0102	ZW0201	WZ0301
TIMER 0	Available for the application	Protocol system clock	Protocol system clock
TIMER 1	Available for the application in case the UART API is not used	Available for the application	Available for the application
TIMER 2	PWM/Timer API	PWM/Timer API	PWM/Timer API
TIMER 3	Protocol system clock	Not available	Not available

Figure 4.8: *The Z-wave Software architecture*

4.5.7 Z-Wave Hardware Interrupts

Figure 4.9 shows the different interrupt calls, according to the Z-Wave modules model. The interrupt calls can be performed in the application layer.

Zw0102	ZW0201	WZ0301
INUM_INT0 INUM_TIMER0 INUM_TIMER1 INUM_TIMER2 INUM_SERIAL0 INUM_ADC	INUM_INT0 INUM_INT1 INUM_TIMER1 INUM_SERIAL INUM_SPI INUM_TRIAC INUM_GP_TIMER INUM_ADC	INUM_INT0 INUM_INT1 INUM_TIMER1 INUM_SERIAL INUM_SPI INUM_TRIAC INUM_GP_TIMER INUM_ADC

Figure 4.9: *The Z-wave Software architecture*

4.6 Z-Wave Protocol

In this section a description of the Z-Wave protocol stack is given. The protocol stack are illustrated in figure 4.10 and contains the following four layers: MAC layer, transfer layer, routing layer and an application layer. Zensys [2007*c*]

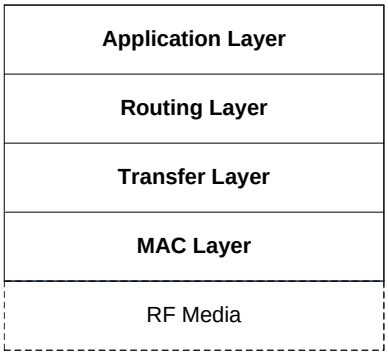

Figure 4.10: *Illustration of the protocol layer on the Z-Wave modules.*

4.6.1 MAC Layer

The MAC layer is the link between the RF media, and the transfer layer. The MAC layer handles the frames from the transfer layer, to the radio frequency media. The Z-Wave data stream consist of a Preamble, Start Of frame, Data and End Of Frame and can be see in figure 4.11. All the data is sent in the little endian format, and the data stream is Manchester coded.

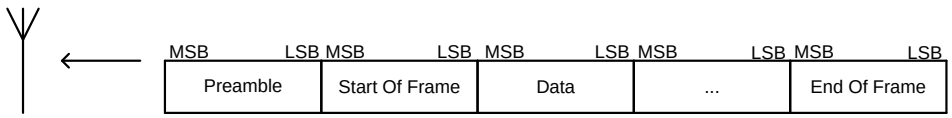

Figure 4.11: *Illustration of the Z-Wave data stream frame.*

The MAC layer has a mechanism that prevents collisions between the nodes, under transmission. The collision avoidance works by letting the nodes be in receive mode when they are not transmitting, and then delay the transmission if the MAC layer is busy. The transmission will be randomly postponed in milliseconds if the media is busy. The time the transmission will be postponed correspond to a random factor, and the time the system uses to send the data frame and receive an acknowledge.

4.6.2 Transfer Layer

The Z-Wave transfer layer handles the data transfer between the nodes, the retransmissions, checksum and acknowledgments. The transfer layer in the Z-Wave modules have 4 basic frames formats, Singlecast, Acknowledge, Multicast and Broadcast frames. The 4 different frames uses the same frame layout which are shown in figure 4.12.

Transfer Acknowledge Frame Type

The acknowledge frame is a singlecast frame with an empty data field .

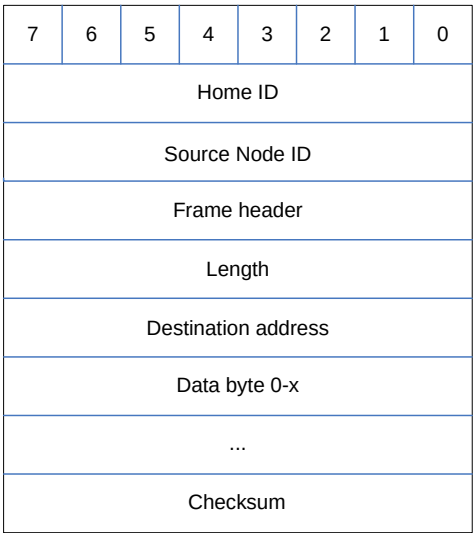

Figure 4.12: *Illustration of the Z-Wave Basic frame layout.*

Singlecast Frame Type

Single cast frames are only transmitted to a single node. If the node receives the frame it will reply with an acknowledge frame. Otherwise, if the transmitted frame is lost or corrupted, the singlecast frame is retransmitted. The retransmission of the frame is randomly delayed, in time steps that correspond to the time the system needs to send the data and receive the acknowledge frame.

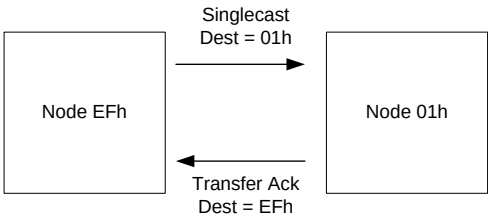

Figure 4.13: *Illustration of the Z-Wave Basic singlecast.*

Multicast Frame Type

The multicast frame makes it possible to send a frame to more than one node at the same time in the ranging from 1 to 232 nodes. This type of transmission do not support acknowledge. It is also possible to transmit a multicast frame to some specific nodes. Due to the acknowledge frames are unsupported, this type of frames can not be used for reliable communication.

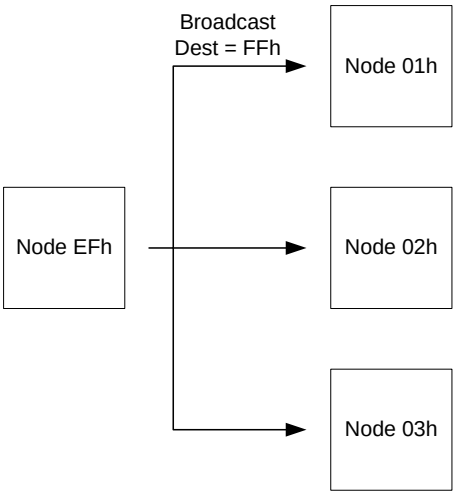

Figure 4.14: *Illustration of the Z-Wave multicast.*

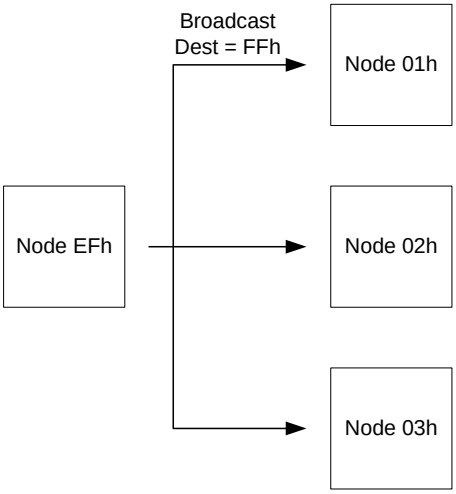

Figure 4.15: *Illustration of the Z-Wave Broadcast.*

Broadcast Frame Type

The broadcast frame is send to all nodes in the network and do not support acknowledge frames. It is not possible to send the broadcast frame to specific nodes.

4.6.3 Routing Layer

The routing layer in the Z-Wave modules handles the routing of frames from between the nodes. The nodes and controllers takes part in the routing of the frames. The modules that participate in the routing are always listening and has a static position. The modules do also scan the network topology and maintaining a routing table in the controller. The routing layer is using two types of frames: Routed Singlecast and Routed Acknowledge type.

Routed Singlecast Frame Type

The controller transmits the frame to one node and the node reply to the controller with an acknowledge. The transmitted frame gets forwarded to the next node and is replied with an acknowledge frame. Using this type of acknowledge frame, the number of acknowledgments the can be kept to a minimal. The controller do only know that the first node have received the frame.

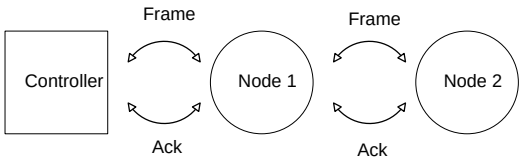

Figure 4.16: *Illustration of the Z-Wave routed singlecast.*

Routed Acknowledge Frame Type

When frames are transmitted using routed acknowledge, the controller is replied with an acknowledge from the receiver. This type of acknowledgment doubles the number of frames which are send in the network. The network traffic can be seen in figure 4.17

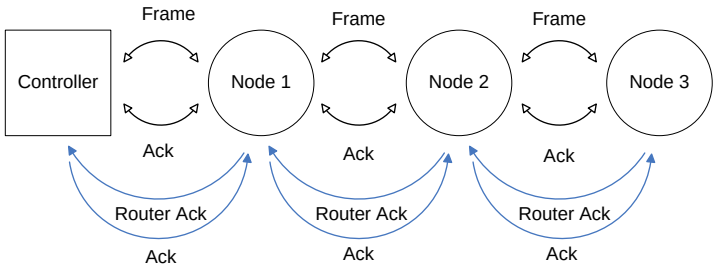

Figure 4.17: *Illustration of the Z-Wave routed singlecast with acknowledge.*

4.6.4 Application Layer

The application layer in the Z-Wave protocol is decoding and executing the commands in the Z-Wave network. The application layer frame holds the Home ID, node ID, application class and application command. The command parameter is sometimes set but no all commands uses more then one command parameter. All the application command classes, application commands and parameters can be found in the Zensys specifications Zensys [2007*d*].

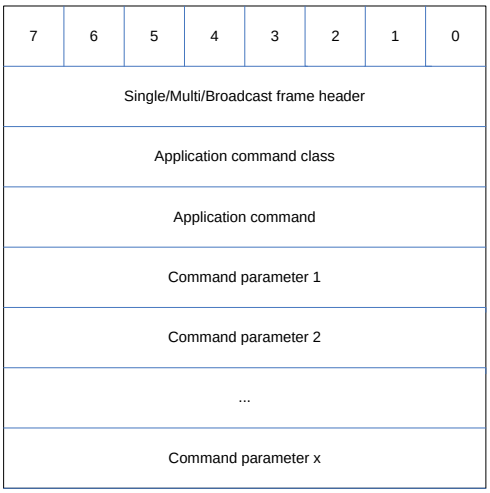

Figure 4.18: *Illustration of the Z-Wave application frame.*

4.7 Middleware

The middleware is the software that facilitate the control architecture, but its more than that. It is a software layer between the application layer and the protocol layer and it can be as extensive as wanted. The middleware is in charge of providing services to the application layer regardless of the underlying network. This requires the middleware to be able to handle a varying number of platforms depending on how complex the system it provides services for is. An example of middleware is the internet, it provides services to different browsers across the globe and together with the TCP/IP protocol ensures that the underlying platforms remains invisble for the end user regardless of platform type [Tanenbaum and van Steen, 2002, p. 36-42]. In this case home automation systems from different manufacturers provide the same functionality but with different and often, by the looks of it, non-compatible platforms. Even manufacturers using the same platform, for instance the Z-Wave platform, have no way of sharing information and services across systems. Ultimately the goal is to design a control architecture that makes it possible for any given application to work on any arbitrary physical network as shown on figure 4.19.

As mentioned several different manufacturers all have successful working systems which can accommodate different aspects of home automation. Getting those manufacturers to change their already working systems is probably not going to work nor is it a very manageble task. Instead, adding a piece of software to each controller in each subsystem which enables them to talk to a primary controller is the idea with this middleware.

4.7.1 Communication Frames

To properly integrate different subsystems into a fully working system requires specified communication frames. The current problem with the different subsystems from different manufacturers is the lack of communication arising from not having a standardized way of doing it. Which means that part of design of the middleware is to specify what information a data package should contain and how to send, receive and interpret the packages sent between the controllers.

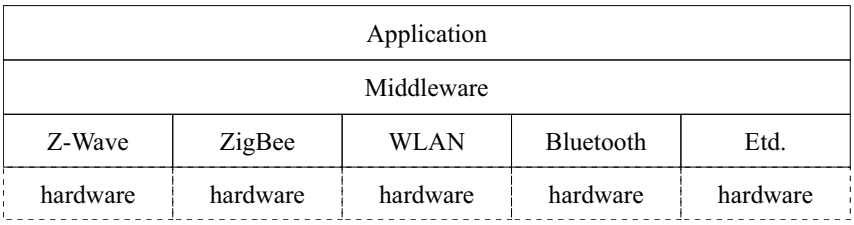

Figure 4.19: *Illustration of the implementation of the middleware between an application and different types of communication protocols.*

The information in such a package sent between two controllers needs to contain all the necessary messages without completely seizing the broadcast channel. To keep the communication short it is important to define what features the information frame should contain. This should also help make sure that all possible scenarios are getting covered so the middleware is capable of handling all different subsystems from different manufacturers. Things to consider are these

- Sender ID. To properly identify the sender the sender ID must be part of message. This is to ensure that both the primary and the secondary controllers are fully aware of where the signal is coming from. If its a temperature for instance, it is also relevant where that temperature is measured.
- Receiver ID. This is a given, not every node needs or can use every piece of information.
- Frame ID. There are different types of packages sent along the network. Acknowledge packages to confirm the reception of other packages to ensure data reliability is a must. There are two different types of data being sent to the nodes, data and parameters, where data is info usually going from the nodes to the controllers, and parameters changes enforced on nodes by controllers to change or optimize their function.
- The actual data. It needs to contain the type of data sent, if its a temperature, if its a boolean variable and the gathered information. The same goes for parameters, the nodes need to know which variable they are changing if they have more than one.
- The length of the data. To make sure the data is received correctly the receiver needs to know how much data is sent.

4.7.2 Middleware Features

For the subsystems to be able to handle adding a new controller to their system, which will be the primary controller, they need additional software. This software needs to be rather complex and flexible, perhaps even specifically tailored to each type of subcontroller, in order to be able to convert the information gathered from the primary controller into something understandable to the subsystem and vice versa. The same type of idea was behind the feature to the Microsoft DirectX when it was developed. The graphic card manufactures could write their software to the directX, which is a collection of API´s, in order to let this directX handle the trafic to e.g a computer game. This is basically the same idea behind this middleware that´s suppose to handle the traffic between the hardware- and application manufactures.

Chapter 5

Requirement Specification

To properly design the middleware a requirement specification is created based on the desired functionality and the demands put forward for the control architecture.

5.1 Delimitation

In the analysis different communication type are described and due to the time available for the project, is it decided to only focus on Z-wave. It is intended for the middleware to contain services such as updating, user interface, storing data and external access using internet, mobile phones and PDA's. The middleware needs to be able to handle larger networks and a multitude of communication platforms, however, this project is delimited to only focus on

- Z-Wave
- Basic communication functions through the middleware

5.2 Requirements

5.2.1 The Control Architecture

The control architecture defines how decisions are made and communicated through to the node doing the task. The cotrol architecture needs

- The ability to pass commands on to the correct subsystems.
- The ability to provide services from the subsystem to the application layer.
- To have limited impact on already existing subsystems.

5.2.2 The Frame Layout

If the communication time needs to be as short as possible, the frame size and the information in the frame need to be precise as possible. The frame frame need to be flexible so it can be used for acknowledge, parameter and data communication.

- The frame must have a destination address, sender address, type, length, data and checksum fields
- The acknowledge does have destination address, sender address, type
- The frame must be general enough to pass on information from a multitude of different manufacturers

5.2.3 Middleware

The middleware is associated with many different subsystems and as such is not necessarily the same on each subsystem. However the properties of the middleware is the same. The middleware needs to

- Define an interface towards the application layer.
- Be able to translate subsystem data into middleware frame data and vice versa.
- Keep track of which subsystem provides which service and keep track of which communication platform is associated with this service

Chapter 6

Middleware and Control Architecture

Based on the requirement specification and the analysis the middleware and control architecture can be designed.

6.1 Choosing Architecture

One of the primary objectives of the control architecture is to provide a transition from the current partly- or non-automated systems to a fully integrated home automation system. With this in mind, a structure has to be chosen which, while still providing the full utility of a complete system, can offer information sharing on whichever level the subsystems are ready to handle it. The advantage of the hierarchical architecture is the minimal influence needed on the already existing and working systems. By adding an additional controller which the subsystems are assigned to, the subsystems can continue their existing functionality while providing extra system wide services at little effort. The primary controller also provides an entry point for a user interface as well as giving easy access to the database, frameworks and applications for upgrades and additions.

The downside off the hierarchical structure is its single point of failure, the primary controller, whereas the distributed system will keep functioning even if it looses the functionality of the failed subsystem. However, the distributed system will have to have a databases added to every subsystem, and to keep this fully updated will require more communication straining the bus. Besides, implementing a functionality where a master role is passed on between different controllers to ensure that no control conflict will occur is an unnecessary complication.

A completely flat structure will demand much uptime and memory from every node, which is not an option with the battery driven Z-Wave modules if they are to preserve some longevity. Besides, this structure will require a completely restructuring of the already existing systems, which will make it relatively harder to propagate the system to manufacturers with a working product.

The hierarchical control architecture is chosen. This type of structure introduces the fewest

problems with the limited data flow available on the Z-Wave platform as well as adding fewest complications to the already existing systems.

Because a hierarchical structure is chosen a primary controller needs to be added to every system with 2 or more subsystems. This will also require additional software added to the subsidiary controllers to facilitate the communication upwards in the hierarchy.

An illustration of the hierarchical communication structure can been seen on figure 6.1. The figure shows a system of two subsystems with subcontrollers and a primary controller. The subsystems have attached nodes which provides information for that subsystem which in turn can provide services for the primary controller using the middleware for communication.

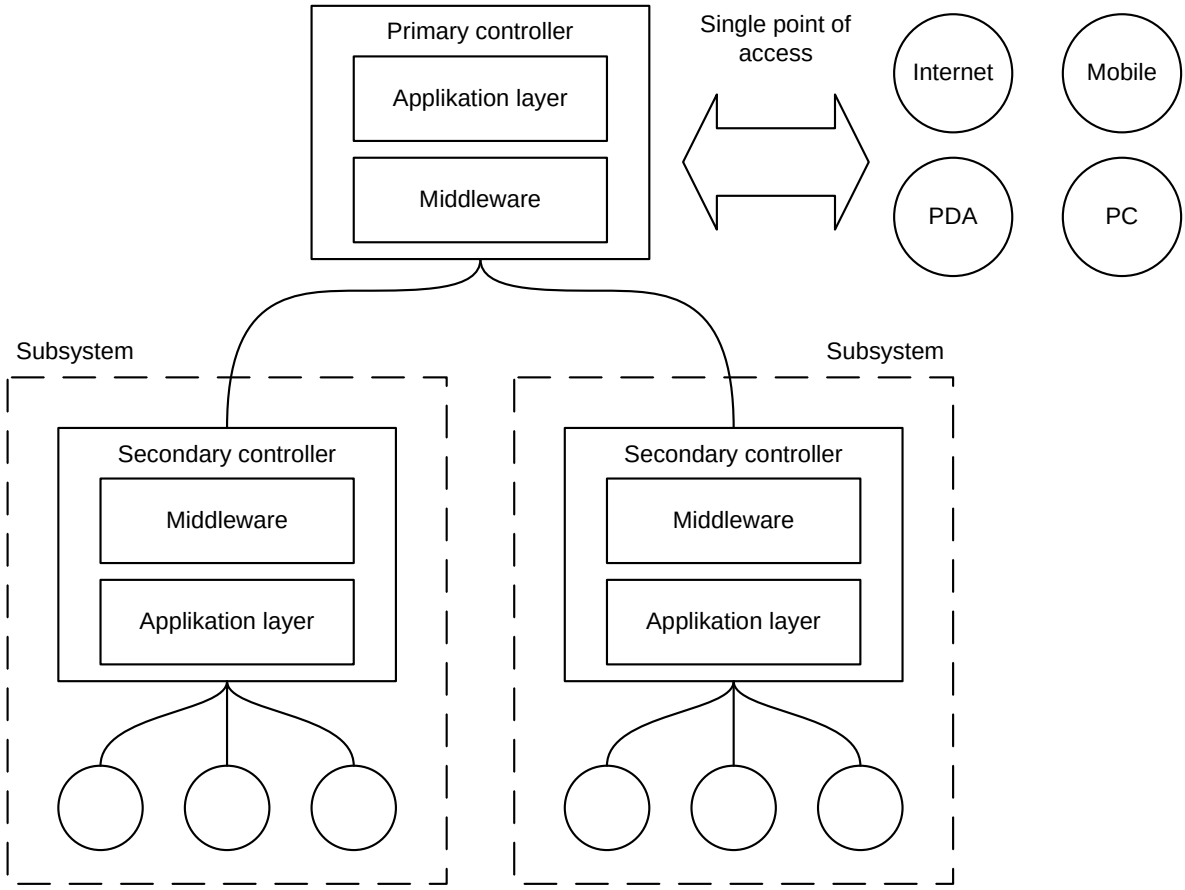

Figure 6.1: *Multiple subsystem in the hierarchical control architecture.*

6.2 Design of Middleware

Specifying a standardized communication frame between subsystems and the primary controller, ensures that information is interpreted by the applications identically and every manufacturer have a set of interfaces to tune their networks to. If done correctly, every different piece of information produced or needed by any arbitrary subsystem should fit into this frame. This way the systems gets the widest flexibility with already existing systems in mind. Then an interface

between the middleware and the subsystems has be defined and implemented in the subcontroller as well. This interface needs to handle the conversion between the communication used by the subsystem and the communication frame specified to handle the communication towards the primary controller. In the primary controller software is added which provides an interface with the application layer and specifies how applications access the information and parameters in the system. Thus the middleware provides the missing link between the individual subsystems and the system wide application layer.

The middleware described in section 4.7 on page 22 needs to be designed. The encapsulation through the layers is illustrated on figure 6.2.

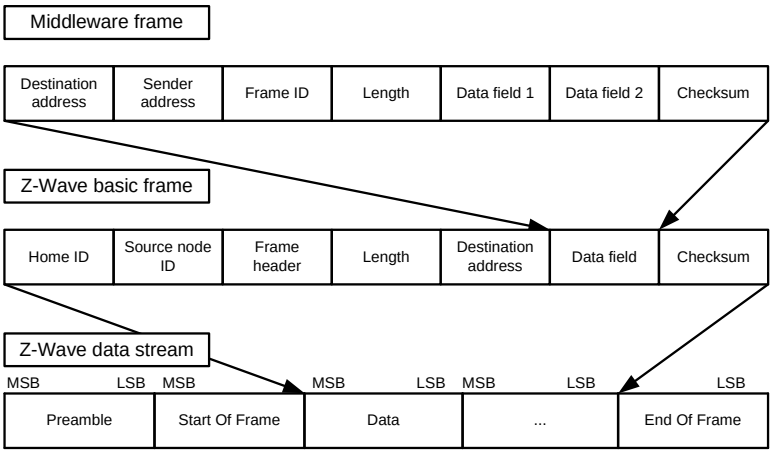

Figure 6.2: *Figure to show the encapsulation of frames from the middleware through the Z-Wave protocol.*

Limiting the system to only work on the Z-Wave platform simplifies the encapsulation process.

6.2.1 The Subcontroller Interface

The software added to the subcontroller varies depends on how the subsystem already processes information. When working with the Z-Wave platform on both the subsystem and the added primary controller, the frame encapsulation needed is not as complex as if the platforms were different. The services offered by a subsystem depends on what the manufacturer of that subsystem put at the primary controllers disposal and this have an effect on the software.

The interface between the middleware and the subsystem needs to contain the ability to encapsulate the frame from the subsystem. The subcontroller also needs software that distinguishes between the information it receives from its subjects, as not all information is relevant outside the subsystem. This selection process needs to be made by the manufacturer of said subsystem, and is possibly already done for most of the systems as most of them already have parameters that can be changed in order to affect functioning of the system. If not, the selection process should consider

- The value of the information for the rest of system.
- Repeated measurements/changes, how often should they be submitted without straining the channel.

- How much control the primary controller should be able to get over the secondary controller.

Design of the middleware part placed on the secondary nodes involves an encapsulation process of the middleware frame in the communication fram used by the communication platform on that subsystem.

The functionality of the subsystems is not touched and is left to the individual manufacturer to maintain. Instead the subcontroller software is extended to also contain software that translates the information from the subsystem into the specified communication frame and vice versa.

For a fully integrated home automation system, all the features of every subsystem needs to be available for the primary controller to understand and influence. The first step however is not to make everything available, it is instead to make a few features available with information which will benefit other parts of the system.

6.2.2 The Primary Controller

The middleware associated with the primary controller provides the interface towards the application layer. Based on the data received a variety of variables and parameters is made available. The subsystems provide a set of services that needs to be available for the application layer. This can be handled in two ways, either by specifically telling the primary controller which services become available after a subsystems has been added. The other approach is making the middleware intelligent enough to recognize services whenever a subsystem is added to the network. The latter is preferable, but it requires an extensive knowledge of available services from different manufacturers. Information such as type of data, type of variable and a value for the variable are all parameters that needs to be accessed and as such needs to be recognized by an intelligent system.

With all the services available systemwide applications can be implemented and by using the middleware, these applications can access information and change parameters in the subsystems. One service could be allowing the primary controller to change a parameter, for instance a temperature setpoint in a subsystem. This could be useful if the primary knows a window have been opened or that the household is on vacation.

The primary controller needs to consist of an application interface, a way for the application to access the services made available by subsystems. The communication platform assoiciated with a certain service needs to be tracked in the middleware as well as the ID of the secondary controller. This way, a service request can be sent using the correct communication platform and to the correct subcontroller providing the service.

6.2.3 Communication Frames

The finished frame layout is showed in figure 6.3. The frame is designed to be flexible and usable, in every way. The frame standard have Destination address, Sender address, Middleware Frame ID, Length, Data fields and Checksum.

Some of the fields in the frame is not necessary if a Z-Wave platform is used, the checksum will be performed on the Z-Wave frame where the middleware frame is located in the data field. This results in a double data check. But if the middleware is to be used by many different platforms, both checksum and acknowledgment is necessary to ensure data reliability, as its not guaranteed that every communication platform contains these features.

Type Frame	Destination address	Sender address	Frame type	Length	Data fields size (X) + 1 byte		Checksum
Data	Unit ID	Unit ID	1	X * 1 byte + 1byte	Type	1 byte	Yes
Parameter	Unit ID	Unit ID	2	2	Type	1 byte	Yes
Acknowledge	Unit ID	Unit ID	3	None	None	None	None

Figure 6.3: *The proposed communication frame*

- The Destination address is located in the front of the frame, and shows which address the frame is to.
- Sender address is used by the receiver, that way the sender can get an acknowledge or the requested data.
- Middleware frame ID is an indication of what the frame contains, according to the value in that field the receiver knows what to do with the frame.
- Length field show the size of the rest of the frame.
- The data fields have the data the sender needs to deliver to the receiver.
- The Checksum shows if the frame is received correctly

6.2.4 Encapsulation of Frames

The encapsulation of frames is handled by the secondary controllers and the middleware controller. Each subsystem runs their own application under supervision from the secondary controller. When interchange of information between the subsystems is needed, the middleware creates the middleware communication frame and sends it down to the subsystem frames data field. Then the subsystem can send it to the middleware controller which then decode it. The encapsulation from the middleware frame to the Z-Wave frame is illustrated in figure 6.4.

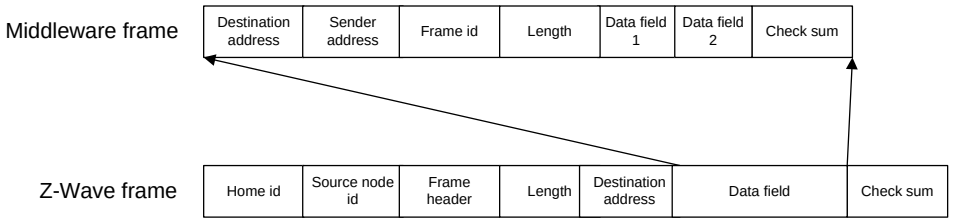

Figure 6.4: *Illustration of the encapsulation of a middleware frame in to a Z-Wave frame.*

The full middleware frame placed in the data field in the Z-Wave frame, then the Z-Wave frame is transmitted to its destination. In the middleware frame the controller is able to open the frame and use the information.

6.3 Protocol Performance

The Z-Wave platform is having a limited data transfer rate. Therefore an investigation is made in order to show how encapsulation of a middleware frame affects performance in the network.

Figure 6.5 shows the Z-Wave frame with and without middleware frame, consisting of header, data and checksum. Encapsulation of the middleware frame entails an increasing header field of 4 bytes. The maximum length of data specified by Zensys is 47 bytes. Including the middleware frame the length of data can be 43 bytes at maximum.

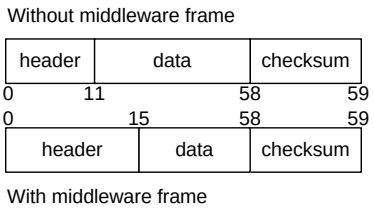

Figure 6.5: *With and without encapsulation of the middleware frame.*

One measure of the performance of a protocol is the goodput, defined as, the amount of useful data from the application level divided by the time to send the frame, including both data and overhead. The goodput is defined in equation (6.1).

$$\text{goodput} = \frac{d_{\text{useful}}}{t} \quad [\text{kbps}] \tag{6.1}$$

where:
d_{useful} is the Z-Wave information without overhead.
t is the transfer time of a frame.

To investigate how data size effects the goodput, calculations for values between 4 and 47 bytes are made for the Z-Wave frame. The last 4 calculations are left out on the graph. Same principle is done when encapsulating the middleware frame, but calculated for values between 4 and 43 bytes. The calculation is based on the mean transfer times given in the Z-Wave specifications with a data rate of 40 kbps. The graph on figure 6.6 shows the goodput as a function of data length.

The difference in goodput for the two plots increases as a consequence of greater data length. However, the differences are in between 5,5 % and 4 % for small and great data lengths, respectivly.

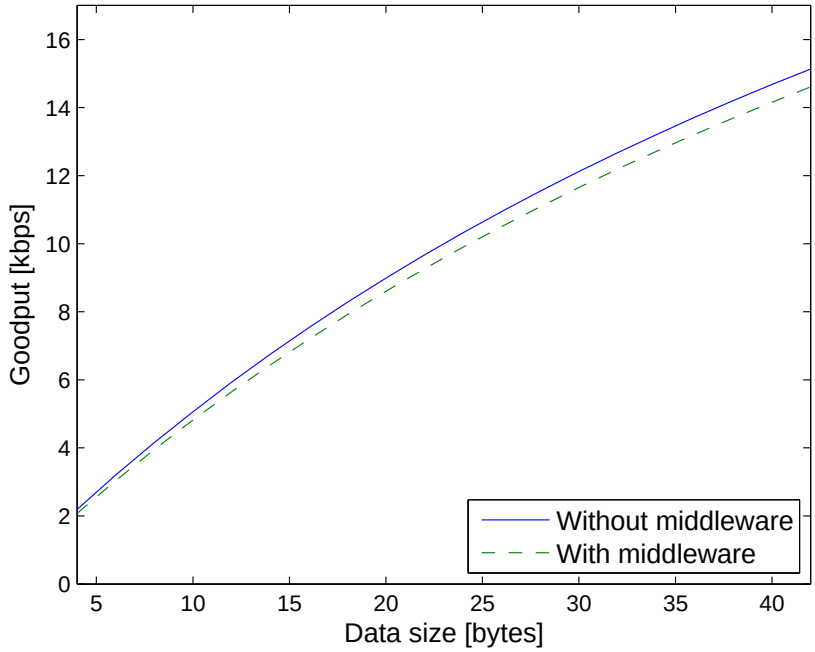

Figure 6.6: *Goodput as a function of the data length.*

Chapter 7

Platform Simulation

The Z-Wave hardware delivered by danfoss was set up in a network by the following PC Controller software. Furthermore the PC Controller application code on the enclosed CD was build in Microsoft Visual C# in order to locate the communication commands between the Z-Wave nodes. The documentation on the CD did not contain any examples on how to use C# as communication to Z-Wave nodes, and therefore it was a very time-consuming process. It was concluded from a simple connection between to computers each with a Z-Wave node connected, that it would be better to simulate the network in order to gather information of a larger network.

The Z-Wave hardware per se is a flat structure. When the basic communication software from Zensys is implemented on the hardware it becomes a hierarchic structure. The platform simulation is decided to contain both the properties from the hardware and software from Zensys. The aim is to design a software platform that can simulate the Z-Wave nodes with the communication software installed. The simulated platform is reduced to only include one primary, two secondary controllers and six slaves. An illustration of the network can be seen on figure 4.3 on page 10.

The purpose of the simulation is to make it possible to do a mapping of the middleware on top of the platform simulation. The ideal situation will be if it is possible to implement the middleware directly on top of the physical hardware.

The hardware is chosen to be programmed and the group has chosen to do this as object oriented programming. This approach has the advantages that classes can be made and then make some inheritances from the classes in order to create objects.

The object oriented approach can be done by using one of different types of programming language. It is chosen to do the programming in C# as the programming language and programming tool makes it possible to make a visual overview of code.

7.1 Analysis of the Platform Simulation

Before the design of the simulation, an analysis is needed. In order to find out how the program should behave, all things related to the program must be described. A brainstorm from figure 4.3 on page 10 all nouns related to the program are found (including their actions and properties). The actions and properties are often verbs and adjectives, respectively.

Nouns:
Node
Primary Controller
Secondary
Slave

Actions/verbs:
Sending

Properties/adjectives:
nodeId
homeId
connectedFrom
connectedTo
powersupply
ZWaveFrame
ZWaveAcknowledge
MiddlewareFrame
MiddlewareAcknowledge
Status

7.2 Design of the Platform Simulation

As showed on figure 7.1 which is a continuation of figure 6.2, the two subsystems are connected by the the primary controller. The frames sent between slave and secondary controllers is developed by Zensys. These frames are processed in secondary controller application. An encapsulated of the middleware frame into a Z-Wave frame is done if the frame needs to be sent to the primary controller.

Even though the whole system contains of both platform simulation and middleware it is based on the same design. From the analysis it is now possible to identify how the program can be implemented. The nouns, verbs and adjectives are often analog to classes, methods and properties, respectively. But it is only a similarity which has no restrictions. From the figure 7.2 the class diagram based on the observations in the previous section are showed.

Since all the Z-Wave nodes are of the same type of hardware it is obvious to define the first class Node. The Node class contains all properties from the previous section. The primary controller inherit from the Node class. The secondary controllers inherits from the primary class, and slaves inherit from secondary class. That means that all nodes can have the properties in the Node class. When objects are created, the constructors are called and initializes the objects.

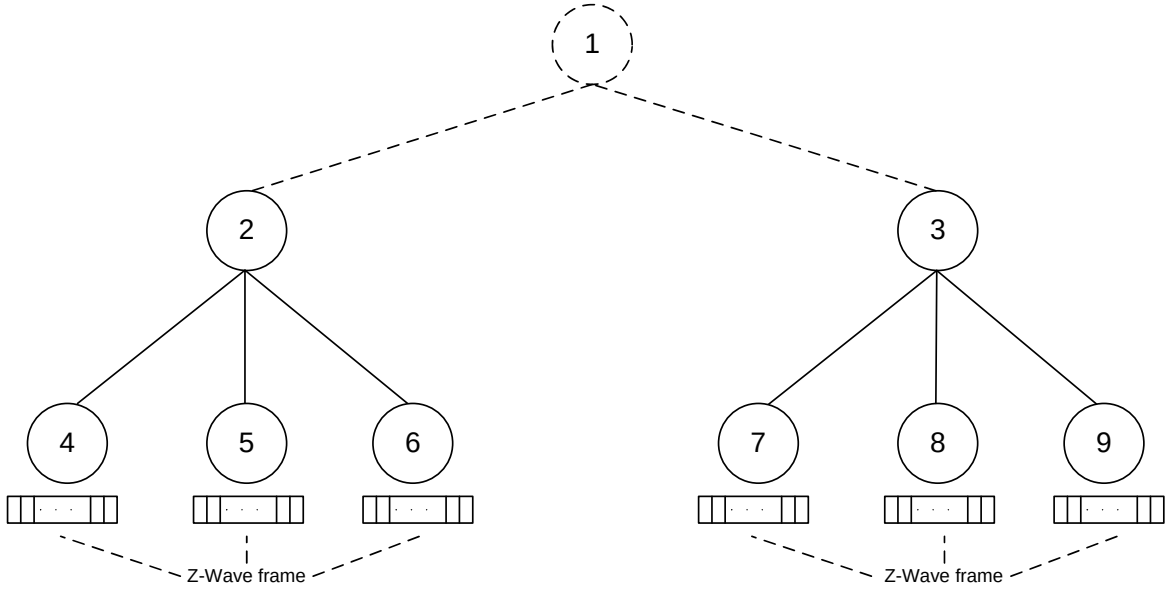

Figure 7.1: *Tree structure of the hardware simulation*

The object orientated structure is not completely utilized since the node objects in the platform simulation is chosen to contain exact values. E.g. on figure 7.1 node 2 is always connected from node 1, and always connected to node 4. This entails that the platform simulation looses its dynamic, if nodes needs to be removed or added later on, but since focus is middleware this is out of the scope.

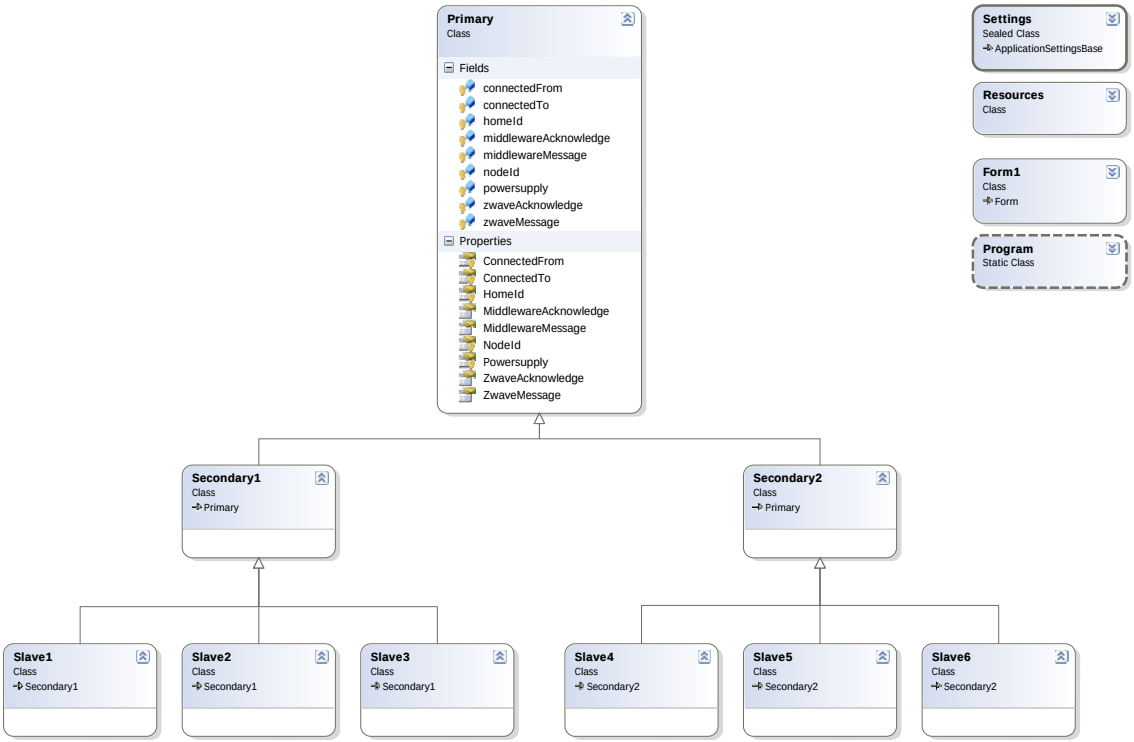

Figure 7.2: *Class diagram of hardware simulation*

7.3 Implementation of the Code

When node objects are created the constructor is called because of the keyword `new`, in order to initialize the objects. Depending on which type of node, and thereby which type of constructor is called, a node id, home id etc. is set up.

To connect nodes so frames can be sent upwards and downwards, the hierarchical communication structure of nodes needs to be set up. The primary controller is not connected from any node but only connected to. Therefore the primary has a connectedTo array of size 2. The same thing is done for the secondary controllers connected to the slaves, each with a size of 3 instead. Every node, except for the primary controller, is connected from one and only one node. This is a value set when each of the relevant objects are created.

7.3.1 Transport of a Frame

In the node class, a field called zwaveFrame is declared as an integer array in order to simulate Z-Wave frames. The Z-Wave frame array contains all the elements needed in the subsystems. The application in the subsystem process this frame and if the primary controller needs to be informed, the result is put in a middleware frame. This frame is then encapsulated into a Z-Wave frame and send to the primary controller. Here an application is evaluating whether this information needs to regulate or inform other subsystems. Here a decapsulation is done after the same principle in order e.g. control of a slave.

The program itself is based on a GUI, see figure 7.3 in where the button "create network" sets up the platform simulation consisting of 1 primary and 2 secondary controllers and 6 slaves representing two subsystems. Furthermore the two subsystems are supposed to share relevant information, which is described in the problem analysis. Entered data can be sent from a given slave in order to reach its destination, and acknowledge confirms the transmission.

Node 9 in the right side of the network is supposed to be connected to a heat pump from an arbitrary manufacturer. Node 4 in the left side on the GUI is connected to a solar panel from another manufacturer. The middleware frame is the link between these subsystems. Applications

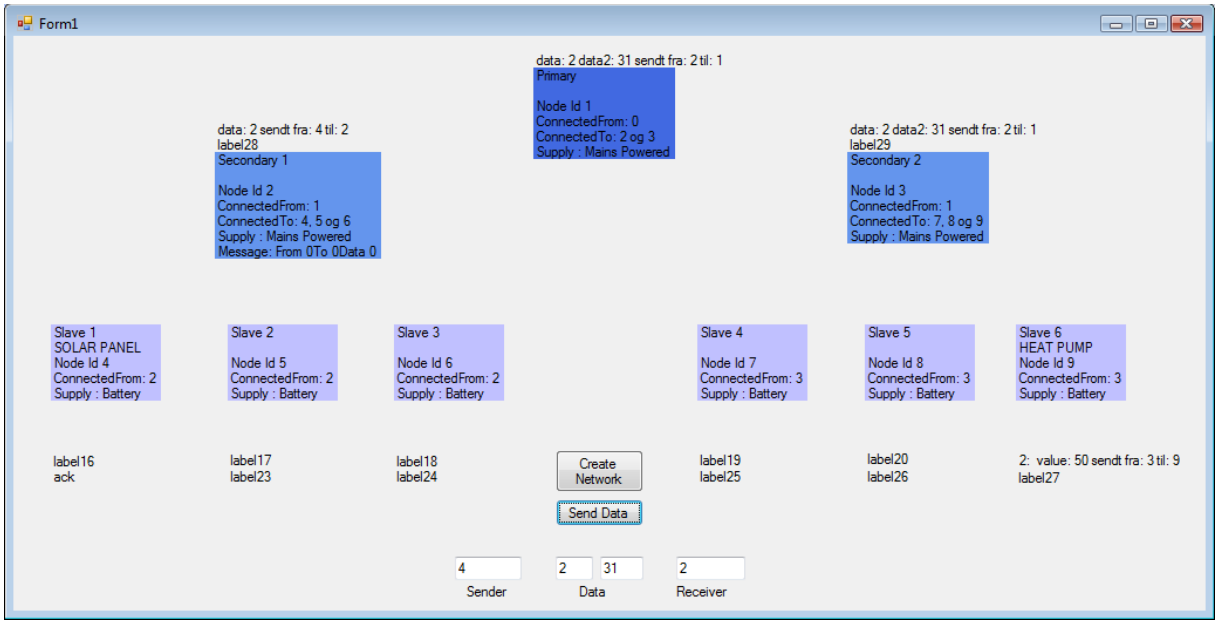

Figure 7.3: *Information from a solar panel is gathered on the left side subsystem and on the behavior of the value it reduces heat pump activity.*

in the primary and secondary controllers are implemented in order to regulate the heat pump based on how much activity there is on the solar panel. Choosing sender node to be 4 and receiver node 2, simulates that the solar panel informs the left side subcontroller whether the solar panel is producing lot of warm water or not. If the entered value in the data field is more than 50, the heat pump is supposed to regulate down. If this value is less than that, the heat pump (node 9) is not suppose to react. If the value is less than 50 and the heat pump is on, nothing is supposed to happened.

Chapter 8

Conclusion

The middleware and control architecture can be applied to numerous systems, however it is a huge area with complex requirements and only part of it has been covered. The hierarchical control architecture provides the most efficient structure for implementing a home automation system, due to minimal impact on existing solutions, while providing the benefit of top-down control and one-point access. Each individual subsystem has their own controller, which manages the applications running in that subsystem, while the primary controller handles system wide applications. This way, existing systems can be integrated in to a complete system, providing the possibility of upgrading to this solution at low cost.

The middleware is responsible for translating the subsystem communication data into the middleware frame format. The middleware in the secondary controllers is designed to access the application in the subsystem, communicating messages to and from the primary controller. On the primary controller the middleware is designed to provide an interface to the system wide applications. The middleware frame was purposely designed to be as flexible and non-complex as possible in order to fit as many manufacturers as possible. Based on the analysis and design it is believed that this has been achieved. If the standardised communication method is followed a link between the controllers is established. Then the integration of the subsystems into a complete home automation system is possible.

A simulation of the physical communication between two subsystem has been performed. The simulation showed that it was possible to get information from two subsystems, through the middleware, utilizing the middleware frame. The simulations shows proof of concept for the middleware communication.

Using the middleware frame in conjunction with the Z-Wave platform has a limited effect on the goodput of the transmission. As Z-Wave is the technology with the lowest bandwidth, the implementation of the middleware frame is assumed possible on all types of examined subsystems, without particularly affecting performance.

Due to the nature and scale of the system, it is not known if the middleware created is able to support all the different manufacturers, and it cannot be determined if the structure chosen will work with all systems, or if the frame supports every data type. The potential of a system

like this is the opportunity to combine specifically tailored systems, already on the market from different manufacturers, covering different areas in home automation, heat, lighting, security, ventilation, etc. This will provide subsystems from manufacturers specialized in their area of expertise.

Bibliography

726, G. [2007], 'Meeting with danfoss on als 10th october', Present: Per Printz Madsen Supervisor AAU, Søren Hansen and Roozbeh Izadi-Zamanabadi Danfoss, group 726 AAU.

Inc, P. M. [2007], Webpage, date: 22 oct.
URL: *http://rfdesign.com/next generation wireless/news/zwave vs zigbee/?cid=zigbee*

Tanenbaum, A. S. and van Steen, M. [2002], *Distributed Systems, principles and paradigms*, Prentice-Hall Inc.

Thomsen, M. K. [2007], 'Kamp om standarder til automatiske hjem', Webpage, date: 23 nov.
URL: *http://ing.dk/artikel/76981*

Walters, M. [2007], Webpage, date: 16 oct.
URL: *http://www.z-wavealliance.org/modules/start/*

Zensys [2007*a*], 'Installation guide', Confidential.

Zensys [2007*b*], 'Programmers guide v5_00', Confidential.

Zensys [2007*c*], 'Z-wave protocol overview sds10243-3', Confidential.

Zensys [2007*d*], 'Zwave_custom_cmd_classes', Confidential.

Appendix A

Scenarios

A.1 Scenario 1

In last section different scenarios where found for each room in the house. The extended scenarios will focus on the house in some states. The states are summer/winter, daytime/evening/nighttime, vacation/not on vacation and someone home/not home/coming home.

There will be 36 different scenarios from the combinations of the 10 state. Instead of descriping all 36 scenarios the focus will be on 2 scenarios.

A.1.1 Scenario 1

The first scenario is in the wintertime and the residents are coming home from vacation in the evening.

Before
Before the residents are arriving in the driveway, the house is in a certain state. All the doors are locked and the security system is enabled. All the lights in the house are off and the light at the front door and in the garden are on. And because the residents are on vacation, the temperature and the ventilation are set to low.

When
When the residents arrives in the driveway a motion sensor detects the car and switches on the light in driveway. From the car the garage door are unlocked and opened and the light in the garage and in the hallway switches on. At the same time the security system is disabled and the front door is unlocked.

The heat system switches from low to normal and the centralized ventilation switches from low to high to distribute more hot air.

After
When the normal temperature is reached, the ventilation swiches to normal.

A.1.2 Scenario 2

The next scenario is in the wintertime and the residents are woken at 7 am in the morning. At 8 am they are leaving for work.

Before
Before the residents wakes up, the house is in night mode. Night mode means that the temperature and ventilation are set to low. Further more the security system is activated to monitoring the doors and windows.

At 6 am the state of the house switches from night mode to day mode. This means that the temperature and ventilation are set to normal.

When
When the residents wakes up the security system deactivates. At the same time the coffee machines starts to make coffee.

After
When the residents leave the house, the security system activates again. If any lights in the house is lit, they will turn off. A timer in the coffee machine is set so that it automatically power off after 60 minutes. The temperature is set to normal and the ventilation system is set to low.

A.2 Scenario 2

Scenarios in the living room and kitchen are chosen to find out which parameters that can affect the temperatures in different ways. This will lead to some functions of the system, described in the next section. The meaning of these functions is to deal with the parameters in order to get more energy efficient and comfortable temperatures.

A.2.1 Living room

Ventilation

A.2.2 Kitchen

A.3 Functions

According to the scenarios it is now suitable to deduce the required functions that the Z-wave modules should have in order to fulfil the From the scenario it was obvious that the sun could have a certain effect on the temperature level in the living room. This leads to a wish for a function that, with a reasonable probability, can predict whether it is necessary to turn the floor heat on, or not. A memory log of temperatures and time is needed for X days, so it is possible to give a qualified guess.

Similar to the living room, the kitchen also have some heat disturbances that can be taken care of by the Z-wave modules. It makes sense if the Z-wave modules do not allow to turn up the floor heat, an hour or two, before the cooker normally is used - even though the temperature is bellow the set point. This compensation for using the cooker is a potential energy save and gives more comfort. Again a memory log of temperatures and time is needed for X days, so it is possible to give a qualified guess.

A.3.1 Possible functions

Functions the system should be able to handle:

- External impacts
  The system should be able to log measurements and based on previous values the system should adjust the heating for similar days. That way the system becomes predictive and can prevent heating overshoot/undershoot because of external heat sources. Gives a more stable temperatur and saves energy. Possible predictions could be
  - Predict/guess the amount of hours the sun is shinning based on the period and the last weeks measurements
  - Predict/guess cooking interval based on the habits
- Timed changes
  The system should be able to adjust heating based on habits. If the inhabitants of the house wake up the same time every morning or the same time on a weekly basis, the system should be able to go from night-time drop to day-time settings before they actually do get up to have the room/house at the right temperature.
- Make a qualified choice when information are conflicting
  If 2 different notes are trying to set a certain temperature then based on a comparison and a predefined algorithm the system should decide which of the set-points should be choosen. This could be done a temporary master/slave condition. One example of this could be 2 different rooms trying to set 2 different temperatures for the heatpump. In this case the heatpump needs to choose the highest temperature and the room with the lowest temperature needs to adjust its heating valve accordingly.
- Accept new notes no matter topology
  If new units are added to the system, after the system detects its purpose, the system should be able to adjust and accept the new inputs/outputs. An example of this could be a new radiator in a house. This radiator should also be allowed to adjust the heatpump if it requires a certain temperature. The system should also accept removal of notes.
- Have failure detection
  If one of the following things happen the system should be able to detect an error and respond accordingly while notifying the user.
  - Be functional even though a note fails or die
    Failure of one note should not cause the system to fail. Even if the system looses a function, the rest of the functions should still be working.

– Be functional even though a actuator/transducer fails or die
  If a transducer/actuator stops working but the note is still working, then the system should not break down just beacuse it cant make that specific unit do what its supposed to