**Project documentation**

**Emergency call button**

Date: **14/02-11**

Advisor: Kim Bjerge

Project participants: Jørgen Vrou Hansen (20042728)

Saiid Shah Alizadeh (19963592)

Anders Hvidgaard Poder (19951439)

Revision: **A**

Document ID: **PRODDOC**

# Introduction

This document details the process, decisions and result of the Emergency button project. A prerequisite for reading this document is to have read the project proposal and the one-page intro.

The document begins by describing the process and then continues to describe the individual parts of the project as it progresses through the process. Naturally the process is iterative in nature, yet in the document any changes will be described under the section they relate to, and the document may therefore not be seen as a chronological detail of the process. Where changes occurs the point of origin will naturally be described along with its affect to earlier section in the change log.

# Process

1. Analyse project proposal and create SRD
   1. Domain model
   2. System level requirements
      1. Use cases.
      2. Sequence diagrams where needed.
      3. Non-functional requirements.
2. Refine SRD to SRS
   1. State and activity diagrams to clarify use cases.
   2. Detailed requirements from system requirements – Should we do this?
   3. Requirement traceability from system to detailed requirements.
   4. Traceability for changed requirements.
3. Overall architectural design
   1. Identify blocks and create overall structure
   2. Mapping of blocks to requirements, both in diagram and RVTM (V is not part of report)
   3. Create internal block diagram for important blocks.
   4. Create activity, state, sequence and other diagrams where needed.
   5. Modify requirements if needed.

3a. SySMl for SoC and design and SystemC transformation

* 1. Relate SySML block to SystemC module
  2. SystemC notation on SysMl diagrams
  3. Partitioning SysML blocks into 1 or more SystemC modules.
  4. Describe the process, important considerations

1. SystemC TLM of overall architectural design
   1. Map functional blocks to SystemC module.
   2. Create communication channels (mostly standard FIFO).
   3. Modify architecture/requirements if needed.
2. Architecture mapping
   1. Identify alternative architectures.
   2. Create architectural design for each alternative.
3. Process mapping for each alternative architecture
   1. Identify processes.
   2. Identify communication.
   3. Map Processes to PE and Communication channels to CE
4. SystemC Timed TLM for each alternative architectures
   1. Update and refine SystemC to the alternative architecture
   2. Identify delayes in the proceses and communication channel based on rough estimation.
   3. Implement delays in SystemC.
   4. Simulate the system and compare the results.
5. Conclusion
   1. Evaluate the pros and cons of the alternative architectures.
   2. Evaluate the process.

# Domain model



Figure 1 - Domain model drawing

In Figure 1 may be seen the overall parts of the Emergency call system including their communication paths. Only a small part of the above is part of this project, yet it is important to realize the domain in which the project operates.

The system functions as follows:

1. An emergency call base is installed in the home of the person receiving care and an emergency call button is issued to that person.
2. The Emergency button communicates with the emergency call base using the ISM network.
3. The emergency call base communicates with the phone company server using its built-in GSM modem and SIM card via a Base Tranceiver Station (BTS).
4. The phone company server forwards the communication via the internet to/from the emergency call server (naturally in a properly protected tunnel).
5. The emergency call server forwards the communication to/from the interested parties; technician, care giver or both.

The above may be formalized in a SysML Domain Basic Block Diagram as shown in Figure 2.



Figure 2 - Emergency call system Domain

In Figure 2 may be seen not only the System Of Interest, but also the environment in which it operates and the blocks with which it interconnects.

The environment in which the emergency call button operates is one where it may be operated by an elderly or disabled person, and may therefore be exposed to some degree of moisture and shocks, as well as worn continuously for a long time.

The ISM network naturally suffers from limited range, noise and interference, which is all part of the ISM network.

The Emergency call base is the communication gateway between the Emergency call server and the Emergency call button. It also has other responsibilities, yet they are irrelevant for this project. The communication goes via the GSM network, which also suffers from limited range, noise and interference as well as a third party service provider (the phone company). In the target area it is believed that with a sufficiently large and well placed GSM antenna a sufficient signal may always be achieved (assumption). Furthermore the GSM network has been thoroughly tested and the service providers have strong incentives to keep a very high availability.

The Emergency call server is placed in a protected environment with sufficient power and network access, and can reach the technician and caregiver on duty using either the head-office LAN , SMS Gateways and/or other third party services.

Focussing on the System-Of-Interest a set of requirements may be created from the project proposal. As the focus on this project is proof-of-concept some of the detail requirements (e.g. colour, frame IP rating, …) will be excluded just like only the critical requirements, especially those containing implemtational risks, will be broken down in the architecture.

# Requirements

This section contains both the overall system requirements (SRD) and the refined requirements (SRS) in one, as the project is an in-house project with close proximity to the “customer”. It therefore does not make sense to create two separate documents, but is far more efficient to simply create the SRS right away.

Again the requirements are only the system requirements and do not include all detailed requirements, yet should contain sufficient data to complete a proof-of-concept.

Before we dive into the creation of the requirements we should look at the Quality attributes of importance to this system, as well as the challenges in meeting these requirements.

## Quality attributes

There are several schools of system architecture, many of which define their own set of quality attributes. A well known and recognized set is the one defined by [BASS] as follow:

* Availability
* Modifiability
* Performance
* Security
* Testability
* Usability

Many add an extra quality attribute:

* Safety

And finally there are some business oriented attributes:

* Time to market
* Cost

Naturally all of these are important, but some are more important than others, and some are also more likely to cause trouble. It is therefore important to include a risk assessment in the quality attribute analysis.

Naturally Safety is vitally important, but at the same time as this product focuses almost entirely on communication, most of the safety aspect is contained in Availability, and the only true Safety aspect is pinching ones finger in the button, scratching oneself on the frame or the battery blowing up. Looking at cell phone manufacturers the last safety aspect is actually a real one, especially as the emergency call button might get damp and warn close to the skin. We do however believe that the current EU regulations cover these aspects, and that sufficient experience exists to avoid this problem, and the risk of violating requirements for the safety attribute is low.

Second come Availability. The button must “always” be able to report an emergency (including battery low situations). Naturally there is no such thing as “always”, so an acceptable probability of failure must be agreed to. In the present system there is no fault detection, but the “buttons” are made so simple that a very high Mean-Time-Between-Failure (MTBF) can be achieved. In this must also be included the probability of a failure later in the communication path, but as these units are on a continuous power supply they can maintain a much higher fault detection frequency. It is believed that the probability of mechanical failure (the button gets stuck or the micro switch fails) can be considered negligible, making the state of the microcontroller a sufficiently deep fault detection.

Usability is also vitally important. Here the experience from existing system may be used to determine the optimal size, colour, location and feel of the button itself. Charging the emergency call button is a much more interesting aspect, as the existing “buttons” very rarely needs to have their battery replaced.

Testability is interesting in the sense that it is important to verify especially the Availability requirements, so it should be considered when writing the test cases and also when implementing the product.

Naturally a certain level of Performance is required, but the response times that is expected lies well below the technical capabilities of the technology available, and the risks here are therefore minor.

Modifiability is more of a nice to have as the cost of the individual emergency call button is such that the entire button may simply be replaced in lieu of updating the firmware, and also gaining access to the buttons themselves are relatively simple, as central lists of their whereabouts are kept at the municipalities.

Finally there is Security, and for this project it is not important. No confidential information should be exchanged on this medium and the desire for someone to want to impersonate an emergency call button is most likely quite small. Naturally the emergency call buttons must be distinguishable from each other, but protection against someone intentionally trying to spoof an emergency call button is not required.

From the above it is possible to create a prioritised list of quality attributes along with their estimated level of complexity (and thereby risk).

|  |  |  |  |
| --- | --- | --- | --- |
| Priority | Quality attribute | Complexity (0 – 10, where 10 is highest complexity, i.e. highest risk) | Flexibility (0 – 10, where 10 is highest flexibility) |
| 1 | Safety | 1 | 0 |
| 2 | Availability | 8 | 3 |
| 3 | Usability | 3 | 2 |
| 4 | Testability | 4 | 8 |
| 5 | Performance | 4 | 5 |

The reason it is important to include the complexity of achieving the required level and the allowed flexibility, is so it is possible to know where there may be some leeway. If the flexibility is 0 it is not even worth looking at, as there are (most likely) strong legislative reasons why this quality attribute must be met to the exact specification. Naturally the complexity and flexibility is a rough estimate based on the developers and architects understanding of the task at hand, and may change as the project progresses, yet if there is a high degree of uncertainty it might be a good idea to reduce this uncertainty before progressing, or at least address this module first in the further development.

When these priorities are in place it is possible to address the business side of the quality attributes. Many newer schools of development argue that you cannot (or at least should not) change the quality, only adjust the number of features included to meet the time and cost schedule. This is often illustrated by the triangle shown in Figure 3, often called the iron triangle because it is possible (ideally) to control two sides of the triangle, but never three. So if you are willing to pay anything, you can have all the features in a very short amount of time, but if you are not willing to pay very much, and you want it yesterday, then you cannot have very many features.

The problem with this triangle is that it does not take non-functional requirements into account, like many of the quality attributes are. The reason this is important is that it may also be possible to save significant money and time if the client is willing to lower the availability requirement, without touching the functional requirements (the features). This is where the complexity comes into play, as it shows how much there might be potentially gained in time and money by adjusting these parameters. This should then be combined with the flexibility, to gain an impression not only on how much is there to be gained, but also how willing (able) is the client to make changes to these requirements.



Figure 3 - Iron triangle

## Rationale

Diving into the actual requirements we can derive a lot of the information directly from the project proposal, current legislation, existing solutions, and some from the specification of the emergency call base. Other requirements, like the maximum allowed time between failure detection, are derived from a combination of risk assessment and acceptable delays as indicated by the caregivers. By estimating a mean time between failures (MTBF) and an acceptable response delay and failure probability it is possible to calculate the required fault detection interval in order to meet the indicated fault probability figure.

This calculation can be done as follow:

* MTBF = 365 days.
* Acceptable extra delay = 30 minutes (a total response time of 60 minutes).
* Acceptable probability that the +30 minutes requirement is not met = 0.1%.

The probability that the error occurs in the time more than 30 minutes from a fault detection is (X – 30) / (365 \* 24\* 60) => (X – 30) / 525600, where X is the interval between fault detection. Combining this with the required probability gives (X – 30) / 525600 = 0.001 => X – 30 = 525.6 => X = 555.6 minutes, or a fault detection every 9 hours, and the2 hours is therefore acceptable.

Naturally the probability that one unit in a collection of units do not meet the above specifications is different (relative to the size of the collection).

## Content

Please refer to [REQSPEC] for the actual requirements.

TODO: Should we insert the requirements here, or refer to the document?

# Architecture

The architecture of the system can be separated into two parts

1. General architecture
2. Mapping specific architecture

The general architecture is characterised by being independent on mapping to HW and SW, where the mapping specific architecture is dependent on selecting the platform on which the general architecture must exist. The Mapping specific architecture is a refinement of the general architecture, and must therefore not be in conflict with the general architecture.

## General architecture

In order to define the general architecture the individual blocks that make up the system of interest must be defined. This is done using a Block Definition Diagram, as may be seen in Figure 3.



Figure 4 - General Architecture Block Definition Diagram

TODO: Is it OK to have blocks that are not owned by anyone???

TODO: What about firmware update?

The creation of Figure 3 comes from analyzing the requirements and grouping the required functionality. It may furthermore be seen that no decision has been made as to what is realized in HW or SW, with the exception of the Housing, which must, for obvious reasons, be realized in HW.

When transforming the requirements into an overall architecture many different techniques may be used. An example is [SOFTARC], and though these books focus on Software Architecture, they may also be for HW/SW architecture. Though many books on architecture exist, a certain level of experience is extremely valuable when choosing the correct architecture.

Looking at the Quality Attributes we can see that it is a relatively static system; no Plug-and-play, no on-the-fly core updates, no third party peripherals. This means that the architecture does not need to include pluggable components or extendable drivers.

The high requirements for availability points to a simple system; the number of errors in a system is always proportional to the amount of code, and also a system that can be tested thoroughly, which requires interfaces for testing the modules independently.

There are many different architectures that can fulfil the requirements, and there may even be several equally good architectures. Therefore choosing an architecture becomes a matter of weighing the quality attributes against the architectural patterns that are suited to satisfy them and combine this with experience. Naturally the architectural patterns should be modified as needed to only include the parts of the patterns that are necessary for this application.

Grouping the requirement into independent blocks using the techniques described above, a possible collection of architecture blocks could be the one shown in Figure 3.

Here is may be seen that the following blocks and their responsibilities has been identified:

* Housing
  + The physical frame that holds the electronics. This is per definition HW, and it has no intelligence or electrical components, only housing.
* Antenna
  + The physical antenna used by the ISM block, as well as potentially any required physical components not part of the ISM block.
* ISM
  + The logic required to package and transmit a data stream / data frame to the Antenna and receive a data stream / data frame from the Antenna. The block is also responsible for maintaining the connection, if there is a connection to maintain (depends on the low-level protocol), required channel hopping, etc. The exact division of responsibility between the Antenna and the ISM block with respect to physical components may be adjusted depending on the mapping.
* LED
  + The physical LED as well as any required low-level driver and physical components required by the LED (often a discrete output is dimensioned so it can drive an LED directly).
* Button
  + The physical button as well as any required low-level driver and physical components required by the button (e.g. a simple hysteresis circuit to prevent multiple activations).
* Battery
  + The physical battery, the charging circuitry and monitoring circuitry as well as any required low-level driver.
* Microphone
  + The physical microphone as well as any required low-level driver and physical components required by the microphone (e.g. an external filtration or amplification circuit).
* Speaker
  + The physical speaker as well as any required low-level driver and physical components required by the speaker (e.g. an external filtration or amplification circuit).
* Control
  + The control logic which maintains the overall state of the system, including timing (when to check battery status, when to send heartbeats), commands from the base (cancel emergency, update firmware) and peripherals (button, LED). The Control logic also is responsible for turning on and off the Audio.
* Communication
  + The communication logic which handles the application level data to and from the ISM block. It is responsible for adding the any required frame header and/or footer specific to the application and to distribute received data to Control or Audio depending on the content (received frame header/footer).
* Signal Strength control
  + The logic responsible for adjusting the transmission strength of the ISM module based on e.g. number of retransmissions (BER), received transmission strength or information from the based.
* Audio
  + The logic responsible for; sampling the microphone, filter out the feedback noise and package the data and send it to the Communication block, receiving audio frames from the communication block, unpack the data and send it to the speaker and the feedback filter. The Audio block may be enabled and disabled depending on whether an emergency is ongoing.

### Dependencies and flow

The above Basic Block Diagram (BBD) only shows the blocks that make up the system and some information about what blocks are part of other blocks, but it does not show how blocks communicate. For that we need the Internal Block Diagram (IBD).

Here we will focus on the important blocks, meaning the blocks that have some data flow, and the blocks that have an interesting interface.

The remaining blocks should naturally also be done, but they are not imperative to do a proof-of-concept or to choose an architecture and platform.

#### Control

The main state machine, as shown in the requirements specification, is implemented in the Control block, and it is therefore interesting to see which interfaces this block exposes and needs. This may be seen in Figure 5.

Here it may also be seen that the all interfaces are standard interfaces, there are no flow data. This is an architectural decision to keep the flow data and the control data separate.



Figure 5 - Control Internal Block Diagram

In Figure 5 the following sub-blocks has been defined with their own responsibilities:

* Command Handler
  + Handle all commands, whether from the communication channel, the button or a timeout of the RTC (heartbeat).
* Communication Handler
  + Handle all communication to and from the Communication block. This block is responsible for parsing the command messages and parsing on the correct command, as well as packaging any responses for transmission.
* Test battery status and perform heartbeat
  + This is the only timer that must be active at all times. It is responsible for waking the system at predefined intervals to send a heartbeat and/or test the battery.

#### Audio

The Audio block is very interesting as it is the block with the most application level real-time processing and filtering.

To specify the inner workings of the Audio block it is necessary to consider the exact division of the responsibility of the Audio block and the speaker and microphone. From the BDD it is indicated that the Microphone and Speaker may contain more than just HW, but it does not say that it has to. Only the Audio block uses the speaker and microphone, so there are no external demands on the separation of responsibility.

As availability (and thereby stability) is very important using tried and proven technology. This goes well with the fact that quality of the audio is not imperative (no need to use state of the art technology to improve sound quality) and with the cost constraint. The battery constraint might warrant considering a non-common design.

In order to make a decision here a small investigative project could be done to determine alternative audio stages and their power requirements, price and maturity. An alternative is to make an educated guess and then make a note that there might be a potential for improving the power consumption, should it later be needed.

Due to the high requirement for availability and the fact that a lot of research has already been done into the power consumption of traditional audio stages, it is believed to be beneficial to postpone the investigation into alternative audio stages.

The traditional implementation of an audio stage is shown in Figure 6.



Figure 6 - Traditional Audio stage

As analog data is subject to noise and loss it is preferable to have the speakers and/or microphone as close to the DAC / ADC as possible. This is the principle used in e.g. active speakers, where the DAC has been moved all the way into the speaker. However for this project the speaker and microphone is already in the immediate vicinity of the Audio block, so there is nothing gained by moving the ADC / DAC out of the Audio block, and this allows a very nice separation into HW and SW, having the speaker and microphone encapsulate everything before the DAC / ADC, and the Audio block contain everything after the DAC / ADC. The DAC / ADC themselves are placed in the Audio block, yet depending on the mapping this might be changed. If the ADC /DAC is an external component it might make more sense to logically see them as HW, where if the ADC / DAC is an integral part of the processing component it might make more sense to model them as part of the Audio block.



There are several decisions that go into this diagram. Firstly the formats of the flow data have been specified; however there is still the matter of how data flows through and between the different internal blocks (the formats only specify what data and the frequency).

There are several different approaches, were the first decision is whether data-loss is allowed. When implementing a system like this there is always at least two clocks; the audio clock (44 kHz) and the main clock (significantly faster). The main clock indicates the speed of calculation and must be able to perform the necessary calculations on a sample before the next sample is ready. This is illustrated on the sequence diagram in



Figure 7 - Audio input processing sequence diagram

In order to evaluate the system’s ability to live up to these timing requirements, there are both external and internal considerations.

As the system is relatively simple and is not required to handle other streams at the same time as the audio stream, it is possible to quite accurately determine the required main clock. The other parts of the system which might consume some processing power while Audio is processing is any commands received on the ISM, heartbeats and battery test as well as button pushes. As Audio is only active during an emergency all these processes (except ISM) can be shut down during an emergency, and the only command that should be received during an emergency is the “Cancel Emergency” command, in which case it is irrelevant whether the Audio processing fails to meet its deadline. It is therefore possible to “simply“ count the required clocks from the ADC is sampled until the data is transmitted on the ISM and add the required clocks from the ISM to the DAC. To this the external factors must be added.

* ISM may lose packages (both ways)
* The may be significant delays in the GSM/ISM network

For lost packages there are two solutions; accepting that they may be lost and continuing (fire and forget) or using an ACK and retransmitting lost packages. The delays are “solved” using buffering. The simplest buffer is 1 frame. This means the buffer only holds what is required to handle one data package, and if the next package is delayed a hole in the sound is experienced. An alternative is to buffer maybe a certain number of ms of data before beginning the playback – effectively introducing a delay. When there is a delay on the transmission line it is usually followed by a burst of data, as all the frames that have been waiting is pushed through. With a single frame buffer only the last package is kept. If there is a buffer it able to smooth out the burst so potentially no data is lost. If the delay is greater than the buffer then data is lost no matter what.

The delay should also be considered, as it should be able to maintain a normal conversation. Naturally much inspiration can be taken from the cell phone industry, and a buffer of 500ms - 1s is a good place to start. Naturally if the implementation phase shows otherwise it may be changed to accommodate.

An obvious place for this buffer is in the audio decoding block. There are two options; Keep the received data buffered and decode it so it fits with the Aduio clock; decode everything into a buffer and simply pop from this buffer using the audio clock. The second solution gives the most deterministic analysis, as the buffer size directly indicate the time delay. If the buffer contains encoded data it is not possible to say exactly how much PCM data there is (without decoding it) – this is naturally only true if an encoding scheme is used that encode certain sounds/frequencies stronger than others. However the first solution requires the least amount of buffer space, as the encoded data always take up less space than the PCM data. By using the minimum required buffer space to keep the desired buffer of encoded data, the smallest possible platform can be chosen, and if a platform with sufficient memory is chosen then the raw buffer solution can always be implemented.

The buffer, whether raw or encoded, must always be circular, so that if a long delay occurs then when the data is burst through, then the oldest data is overridden. Naturally during this long delay the buffer would have been emptied, and though there are algorithms for smoothing out small gaps by using e.g. interpolation and statistical guessing. These algorithms are believed to be unnecessary for this application, and if the buffer is empty a value of 0 is sent to the DAC.

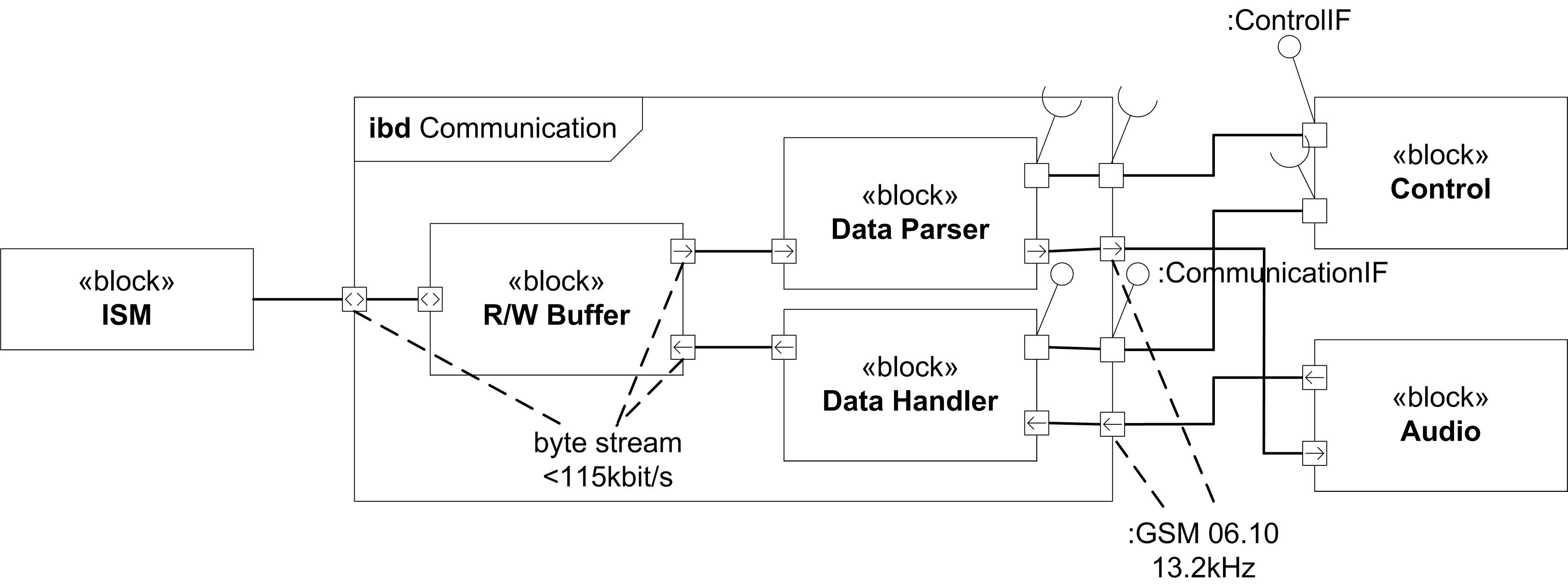
In Figure 7 a section has been left out. It is the secondary input to the feedback filtration. Feedback filtration is simply put just a matter of subtracting the “noise” generated by the speakers from the microphone input. To do this it is required to know the exact delay and transfer function for the system from the data is sent to the DAC until it is pickup up by the ADC. The exact calculation of the transfer function and delay will not be done here, and will be left for the implementation phase. The technology is well known and implemented in any semi-modern cell phone and the algorithms are readily available. The feedback filtration might also need a small buffer, as it may be necessary to keep a certain number of past values, as well as the coefficients for the feedback filtration algorithm. The simplest form of feedback filtration is if the distortion is 0 and the delay is 0. In that case it is simply a matter of subtracting what is played on the speaker from what is received by the microphone. The data flow for the speaker and microphone as it relates to feedback filtration is shown in Figure 8.



Figure 8 - Feedback filtration

As the main clock must always be able to handle one sample in the time before the next sample arrives, no buffers are required to hold the sampled data from the ADC. It is however imperative that there is complete control of the timing in the splitter and feedback filtration.

The simplest is to run both the Audio from the microphone and the audio to the speaker on the same audio clock and design the splitter and feedback filtration so the Feedback filtration always gets the current value or the last value (deterministically). It is important to consider when the feedback filtration request / uses the value from the splitter.





\*\*\*In the heartbeat it is possible to report status information, e.g. battery level.

References:

* Pattern-Oriented Software Architecture