Universidade de Coimbra

Master Thesis

Medical Systems Integration

Author:
Manuel Teixeira Cabeleira

Supervisor:
Paulo Fernando Pereira de Carvalho
Ricardo Manuel Ribeiro do Sal

A thesis submitted in fulfilment of the requirements for the degree of Master in Informatics Engineering

in the

Departamento de Engenharia Informática

August 2014

Abstract

In this thesis is described a system integration solution that can acquire data from medical devices such as, ventilators, infusion pumps and Bispectral index monitors. The system is also capable of providing a broker capable of sharing medical device data with other applications running in the healthcare unit.

The implemented solution intends to tackle the problem of delivering medical data from medical devices to multiple applications with different data requirements. By solving this problem we aim to increase the amount of information available in computer applications of intensive care units.

Acknowledgements

Firstly I would like to thank my supervisors, Eng. Ricardo Sal and Dr. Paulo de Carvalho as without their support and advices, this work could not have been done.

I would also like to thank all my fellow students that helped me with insightful contributions and companionship.

Finally and most importantly I would like to thank my parents António Cabeleira and Maria José Teixeira and my girlfriend Filipa Valente for their constant and unconditional support.

Contents

List of Figures

List of Tables

Abbreviations

ICU	**I**ntensive **C**are **U**nit
BIS	**Bis**pectral index
MD	**M**edical **D**evice
DSS	**D**ecision **S**upport **S**ystem
HSJ	**H**ospital **S**. **J**oão
DEI	**D**epartment of **I**nformatics **E**ngineering
HL7	**H**ealth **L**evel **7**
ANSI	**A**merican **N**ational**S**tandards **I**nstitute
MDF	**M**essage **D**evelopment **F**ramework
UML	**U**nied **M**odeling **L**anguage
RIM	**R**nformation **I**odeling **M**odel
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
ISO	**I**nternational **O**rganisation for **S**tandardization
CEN	**C**omité **E**uropéen de **N**ormalisation
EHR	**E**lectronic **H**ealth **R**ecord
DIM	**D**omain **I**nformation **M**odel
MDDL	**M**edical **D**evice **D**ata**L**anguage

This thesis is dedicated to my parents who have always supported me throughout all this process...

Chapter 1

Introduction and Methodology

1.1 Motivation

The Intensive Care Unit (ICU) of the 21th century is an extremely complex and diverse environment where life or death decisions are made every day. In this environment where huge amounts of data are generated every hour, proper handling of this data has been proven to improve decision making, reduce patient's lengths of stay, increase clinician's productivity and overall service quality and reduce operating costs [1] .

In this work is proposed an integration solution that tackles the problem of sharing Medical Device (MD) data in a transparent way to all the interested parties in the ICU. In order to do this an application was developed that was capable of gathering all data generated by Medical Devices in an ICU, in particular ventilators, infusion pumps and BIS monitors and provide an standard based interface through a gateway to other applications.

This solution will provide a tool to respond to an emerging problem in the data-flow of ICU that is the appearance of applications like Decision Support Systems (DSS) or scientific research applications that will oftentimes "compete" with the central service, which manages the Electronic Patient Record and the unit's workflow, for the access to data from the unit's MDs.

1.2 Framming and Problem Defenition

The neurological patient's ICU service of Porto's Hospital de S. João (HSJ) has a central service solution (B-ICU.Care [2]) and a DSS application (ICM+ [3]) to help the clinicians monitor their patients and improve their decision making.

B-ICU.Care is fully integrated with both the Hospital Information System and the Laboratory Information System and manages all the workflow in that hospital unit. This solution is also responsible of acquiring and storing data from MDs. Due to the storing feature of this solution, the sampling rate at which this data acquired is low and no curve data is never acquired in order to keep the volume of data in the database affordable. B-ICU.Care also has a broker capable of integrating with other applications, but only data stored in the database can be shared.

ICM+ is a clinical research tool used by clinicians to develop and test new indicators to better understand and treat neurological disorders. This application uses data directly acquired from the MDs at high sampling rates. This is a stand-alone application that has limited means of communication data to other applications.

Each of these applications provides useful information to the clinician but at the current state of the unit's information system, they cannot be used simultaneously. This happens because they both use the MDs as source of data and most of the MDs in use at the unit can only provide a single RS-232 channel and therefore can only be monitored by a single application at a time. Even though B-ICU.Care can share its data with ICM+ via broker, this data would not be detailed enough. ICM+ is a research tool, therefore in its construction no importance was given to this subject and a broker to share data was never implemented.

The main purpose of this thesis is to develop a tool to tackle this problem of sharing of data in an ICU unit. For this we propose a Medical Device Integration system capable of integrating all acquired data from the unit's MDs in a standard based broker. This way all current and future applications operating in the ICU will have a simple way of acquiring all the data they need from MDs.

In order to avoid future interoperability problems with other devices or applications, the solution will be developed in a way that allows future developers to quickly add new modules to communicate with other applications and devices that need to be integrated.

1.3 Project Team and Collaborators

The project team is composed by the following elements:

- **Paulo de Carvalho (1)**- Supervisor at DEI
- **Eng. Ricardo Sal (2)**- B-Simple Coordinator
- **Peter Smielewski (3)**- ICM+ Consultant

- **Celeste Dias (4)**- HSJ Consultant
- **Manuel Cabeleira (1)**- Final Project Student

(1) Department of Informatics Engineering (DEI) is the entity responsible for the academic part of the project.

(2) b-Simple is the engeneering company responsible for the project. It is also the owner of the application B-ICU.Care.

(3) University of Cambridge Enterprise is the owner of the application ICM+. It is responsible for adapting the ICM+ application to communicate with the developed solution.

(4) Hospital S. João is the healthcare unit that will be used to test the developed solution.

1.4 Work methodology

During the Requirement analysis and Architecture development stages of the development of this thesis, no access to the applications or to the medical devices was required. As such the work was performed in DEI facilities resorting to a computer with internet access. During the development and testing phases, the work was performed both at the HSJ when access to the equipment was necessary and at home when not.

1.4.1 Software development methodology

The development methodology used in this work was inspired in the iterative and incremental development method, but here the solution only transitioned to production after a final testing phase where in theory the developed solution should have been transitioning to production iteratively. The development work was divided in 4 phases:

- An initial planning phase where a requirement analysis was performed in order to gather all the important information about the project and access the risks involved.

- The second phase was to design a system architecture that would embody all features the developed software should have in order to answer all the gathered requirements.
- In the third phase the application was developed incrementally by starting with a small yet functional and testable system resulting of a small part of the designed architecture. Then continuing adding other parts of the architecture to the system. On each new development, the system was tested for the features it incorporated before adding more content
- After the final prototype was implemented, a final test phase was conducted where all the load and stress tests were performed.

This development method has proven quite convenient because some parts of the project were similar to each other, with this approach by developing and testing one of these parts, the future development of the others proved faster. Debugging was also facilitated as the developed code between tests was small.

1.4.2 Quality Control Mechanisms

To ensure that the final product meets all the listed requisites, the student always validated each document and major breakthrough with the supervisors via weekly 15-5 model reports and monthly face-to-face meetings with the supervisors. Every time something needed to be presented to validation by any of the external stakeholders, for example the person responsible for the applications, the email method was employed.

To improve the quality of the solution, each new development was tested to avoid the propagation of errors during the development. The solution only passed to the production stage after a final testing phase to further guarantee the fulfilment of all the requisites.

In order to keep future developments easy to add, every single part of the project will be documented in this document.

1.4.3 Risk analisys

The development of this project is imbued with several risks:

- The unavailability of the communication protocol document for a MD that needs to be integrated. This can deeply affect the development of the project and in order

to avoid this risk, all the communication protocols have been gathered before the project began.

- Unavailability of MDs to test the solution under development. This risk can sabotage most of the solutions testing phase. In order to minimize this risk, we have contacted HSJ and we got clearance to test the solution in their facilities.
- Delays in the development of the interface from the side of the ICM+ application. This risk can also sabotage the solutions testing phase as it will make impossible to test the solution in a real situation. In order to avoid this problem, a protocol test tool developed by B-Simple company will be used in the testing phase.

At the beginning of the testing phase ICM+ application was still not ready for testing, therefore the solution was only tested by the B-Simple side in a real situation

1.5 Planning and Evolution

The work presented in this thesis was performed in between the sixteenth of September of 2013 and the second of September of 2014.

During the first two weeks of the project some time was taken to study the existing system, the applications B-ICUCare and ICM+ and all the technologies involved in the development of the solution. From this time the first phase of the project was planned and a Gantt was produced (figure 1.1). This first phase covers the first two steps of the software development methodology employed in this work.

The first period of this first phase took roughly 8 weeks and was used to make a study of the state of the art where the student firstly took acquaintance of the medical equipments that were to be integrated in the development solution along with the protocols that those devices use to communicate. A literature survey was also conducted in order to acquire information about the international standards in medical device communication, existing technologies and similar solutions present in the market.

This period was conducted alongside the requirement analysis period where several electronic meetings with the developers of the applications were conducted. One formal session also took place, with Dr. Celeste Dias and Eng. Ricardo Sal. The elaboration of the requirement document was accomplished in one week after the period of the study of the state of the art.

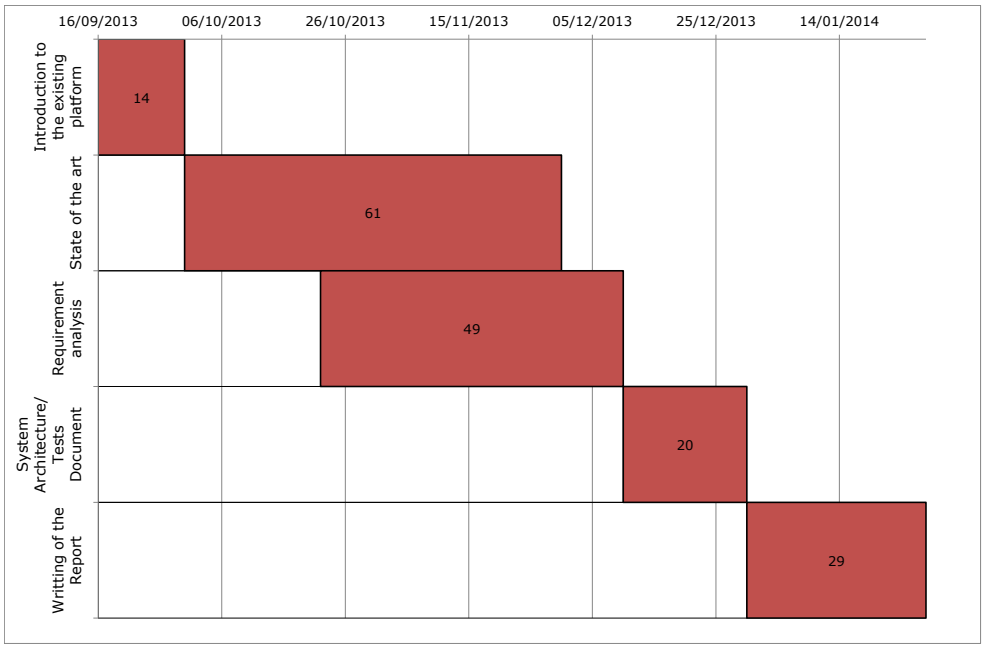

FIGURE 1.1: Gantt of the 1st Semester

The next period of roughly three weeks was used to develop and validate the system architecture. The last phase of 4 weeks was used to write an intermediate report and the Test document.

As can be seen in the Gantt the initial planning was fulfilled with few deviations. The major deviation to the plan was the development of the test document that was performed alongside the period dedicated to writing the intermediate report.

The second phase of the development of this thesis covers the implantation and testing steps of the software development methodology employed. The planning of this phase was performed at the end of the first phase, from this planning the Gant diagram of figure 1.2 was produced.

This plan envisioned a 2 month period to develop the module DataAquisiiton responsible for the acquisition of the data from the MDs.

The second phase of the plan was destined to the development of the ExtAppCommunication module that would implement an Broker capable of transferring data from the MDs.

The third phase was destined to the development of the DataBase module that would save all the acquired data in a shared database with the B-Simple application and would take 9 days to be accomplished.

In the fourth phase the EventLogger module was to be implemented in 5 days where all relevant occurrences would be logged.

The fifth phase of about 2 weeks the final tests would be made to the application In the last month all the documentation would be produced.

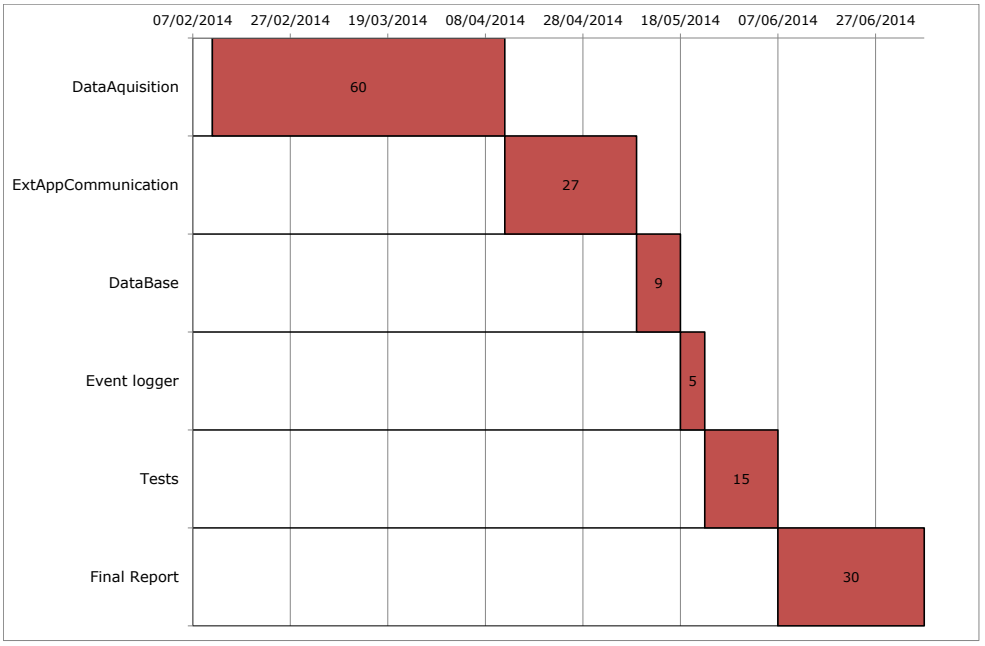

FIGURE 1.2: GANTT for the 2nd Semester

This plan proved however difficult to fulfil due to several factors:

- The expected amount work for the Data Acquisition module was underestimated, mainly due to the complexity of some protocols.
- Scheduling tests in the testing phase also proved difficult to accomplish because we had to test two applications at the same time, due to the need of having to apply the contingency plan of creating a client for our broker in the B-Simple side.
- The plan was also altered during the development time due to the need of applying the contingency plan of developing an application from the b-simple side to serve

as a client to our application's broker, the DataBase module became obsolete and was removed.

- The Event Logger module was found to be implemented alongside with the other phases and therefore was also removed from the Gantt

Because of some of these reasons it was decided to postpone the delivery of this thesis from the early July to early September, in order to be able to accomplish everything in time. In order to compare the initial planning to the actual outcome of the development process another gantt with the actual timings of each of the development phases is presented in figure 1.3

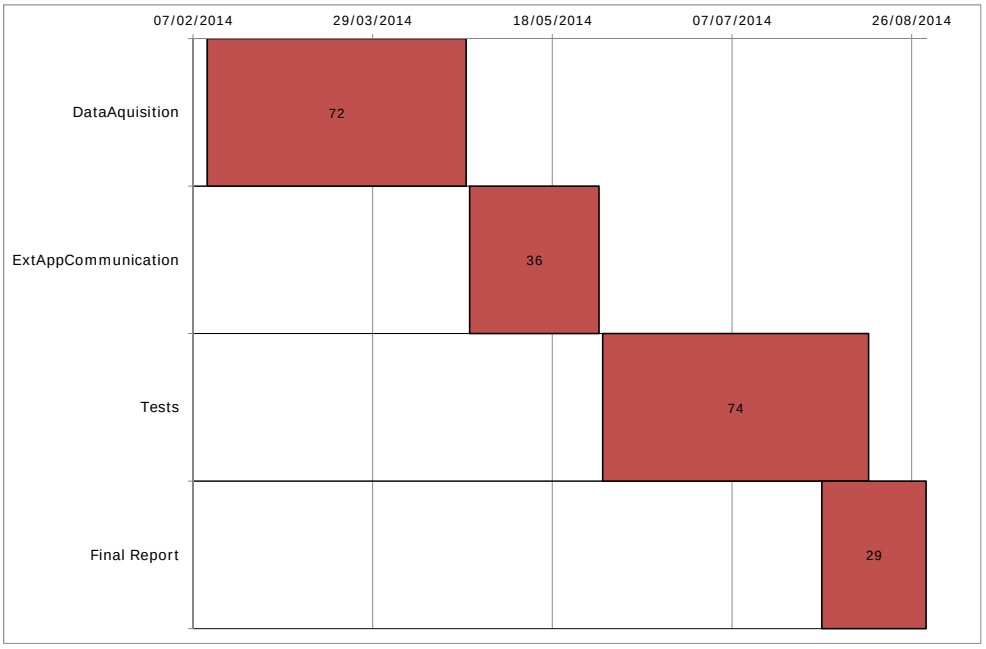

FIGURE 1.3: Final GANTT for the 2nd Semester

Chapter 2

State of the Art

2.1 Introduction

The ICU is a hospital unit where patients are often connected to a wide range of MDs like ventilators, vital signs monitors, infusion pumps, etc. In the last few decades advances in technology have bestowed the ICU with MDs capable of providing huge amounts of valuable information to the clinicians, but each device uses its own communication protocol developed by the device's manufacturer. These protocols are often proprietary and can only work alone or inside single-vendor equipment. Even though this situation creates many interoperability issues, the MD's manufacturing companies are not interested in interoperating with each other due to several reasons. The first reason is the lack of regulatory requirements regarding interoperability. Implementing this kind of regulation is hard, because if a MD's purpose was also to communicate data to other systems, the manufacturers would have to test the device with every generic system to ensure safety and effectiveness generating unaffordable financial costs [4]. In this scenario it would also be impossible to ensure compatibility with future devices. Another reason is customer retention. MD market is highly competitive and if a device has a company specific communication protocol it becomes harder to a client to change to a device of another company.

In this scenario several organizations have tackled this problem by developing standards for the interoperability of medical information systems. In regard to the ICU there are two standards that are well known and widely accepted, the Health Level 7 (HL7) and the ISO/IEEE 11073.

2.2 HL7

Health Level 7 is the name of an organization certified by the ANSI as a standard developing organization committed solely to the healthcare domain [5]. This organization focuses on the production of standards to provide interoperability in the data exchange processes for all the healthcare system, including sectors like administration and finances. The standards developed by this organization with more success are the messaging protocols Hl7 v2 and HL7 v3.

This organization was created in 1987, during this time and throughout most of the 90's the interfaces between systems in healthcare units were very limited and costly because each interface was custom designed, requiring extensive programming by all the parts involved. In this scope the first versions of the standard were developed by clinical interface specialists. These developers noted that in order to drastically reduce the interfacing costs it was only necessary to predefine 80% of the interface framework upfront. The remaining 20% could be custom made by a given facility to reflect special cases.

The first usable version of the standard was released in 1990 by the name of HL7 v2.1. This version and its successor, HL7 v2.2 (1994), were vague and under documented in order to make this standard flexible enough to attract as many users as possible. The user base of this standard kept growing and new versions of the standard were released accordingly, HL7 v.2.3 (1997) and HL7 v.2.3.1 (1999). By this point the acceptance level of this standard was so high that almost every new application in the healthcare area was compliant with the standard. Four more versions were released until the present, the version HL7 v.2.4 (2000), HL7 v.2.5 (2003), HL7 v.2.5.1 (2007) and the HL7 v.2.6 (2008). All these versions are backward compatible, making an application developed in the version 2.3 able to process messages for the 2.2 version. The next section summarizes the HL7 v2.x message protocol.

2.2.1 HL7 version 2.x

All v.2.x messages are encoded as ASCII text strings with delimiters and constitute the atomic unit of data transferred between systems. Each message is always initialized with an ACSII character '0x0b' and terminated by an '0x1c' character followed by a carriage return character.

The set of documents that constitute the HL7 v.2.x define a base message structure that can be used to transport different types of messages, it can produce messages relative to, patient management, financial services, laboratory tests, etc. These messages can

also come in the form of queries, results, observations, etc. In order to compose these messages the standard defines different segments and a valid combination of segments makes a message.

- Segments

The body of a message is composed of segments and a message has an indeterminate number of segments. Depending of the message type a given segment may be mandatory or optional, it can also be repeatable or not. The standard also allows Z segments, these segments are completely customizable, but its use must be considered carefully. Each segment defined has a unique three characters identifier and it logically groups data, for example the PID (patient identification) segment contains data related to a patient, like his name and address.

- Fields

Each segment of a message is composed by a given set of fields. Each field is composed by a size variable string and is delimited by a '|' character. A valid segment must contain all the fields that define them delimited, because each field contains semantics in the scope of the standard and a field is identified by its position in the message. There are three different data types to define a field. They can be defined internally, using a table of the standard, externally, using a vocabulary like ICD (International Classification of Diseases [6]) or by the user in the Z segments. A field can also be single or composed. An element of a composed field can be separated by using the character '^' as separator. The subcomponent separator is the character '&' and the character '~' can be used to repeat a different version of a component.

- Example message

In figure 2.1 an example message is presented. This message is referent to an unsolicited observation result (ORU) composed of 9 different segments. The first segment is the Message header (MSH) and contains information about the sending application (B-Sharer), the receiving application (BICUCare), the message type (ORU^R01), the message ID (1408480019815) and the HL7 version number (2.4). The second segment is referent to the Patient Identification (PID) and fields with the patient internal ID (1), the patient name (Smith^John). The following segment is the patient visit segment (PV1) containing a field relative to the bed where the patient is (^^2). The Segment (PV2) contains more information about the patients loation. The next segment is the

Observation segment that contains the date of the observation (20140819202654). This segment is used to provide generic information about the following Observation fields and can have associated to it numerous Observation segments. In the Observation segments are inserted fields with the actual results of the observation. The first observation contains a field with the Value Type (NM abbreviation for 'numeric'), the second field is the Observation identifier (280ˆTVexp) the next field is the actual value of the observation and is followed by the field where the units of the measurement are present, in this case is a ml (mililiter). The other OBR segments follow the same logic.

```
MSH|^~\&|B-Sharer|HSJ|BICUCare|HSJ|20140819202700||ORUˆR01|1408480019815|P|2.4||1
PID|||2||SmithˆJohn
PV1||I|^^2
PV2|Neurocríticosˆ ICUˆ2ˆHSJ
OBR|1|||ˆMedibusEvita4|||20140819202654
OBX|1|NM|280ˆTVexp||600|mL|||||F
OBX|2|NM|118ˆMVexp||9.5|L/min|||||F
OBX|3|NM|774ˆFR||16|1/min|||||F
OBX|4|NM|594ˆSpRR||0|1/min|||||F
```

FIGURE 2.1: Hl7 Message example

- HL7 v.2.X Limitations

The objectives of the older 2.x version were pretty much achieved, because they successfully reduced the costs of interfacing systems and created a common language the most of the healthcare systems speak nowadays. But now that the system is widely accepted, the ambiguity and flexibility imbued in the protocol are now the main sources of criticism in this standard, being often framed as the 'non-standard standard'.

This flexibility is now the standards main weakness because, even though the standard defines restrictions to the composition of the segments and the fields, it is still possible to implement custom messages that violate the standard if the changes are previously negotiated between the implementing parties. This situation happens regularly and causes each interface to still require special attention.

In order to work around this problem, the HL7 organization has developed a new version of the standard, the HL7 v3 [7]. This new version defines a more consistent data model and is less flexible. It was also decided that the new standard would not be compatible with the older versions transforming it into a new standard. This new version has been

released in late 2005 and promises to predefine 90% of the interface. The next section summarizes the HL7 v.3 message protocol [8].

2.2.2 HL7 v.3

For this new protocol, HL7 has replaced the ASCII messages with an object oriented approach that exchanges XML messages. This version makes use of the new developed Message Development Framework (MDF) that makes use of Unified Modeling Language (UML) design concepts.

- Reference Information Model (RIM)

The RIM is the cornerstone of the HL7 v.3 information models. It is the 'grammar' of the standard and specifies a set of 'building blocks' that imbue the data in a message with semantic context, and lexical connection with other fields of the message. The Rim consists of six core classes:

- Act: In this standard every happening is an Act and therefore this class represents the actions that are executed in the healthcare domain. This class has several specializations, for example, an observation or a procedure.
- Entity: Every Act involves interaction of entities, therefore this class represents every object or being related to an Act. This class also has several specializations, for example, an entity can be a living subject or a material or even a place.
- Role: Every entity performs a role, in this class is represented the role that an entity plays in an act. There are also several specializations for role, for example, if the entity is a person, the role can be 'patient' or 'nurse'.
- ActRelationship: Represents the links or relationships between Acts
- Participation: Represents the relation and involvement between a Role and an Act
- RoleLink: Represents the links or relationships between Roles

In figure 2.2 a model of the RIM is presented. Each class of the RIM has a pre-defined set of attributes and each attribute has a predefined data type. These attributes and data types are then used in the XML messages as tags. HL7 v.3 supports several data types:

- Basic data types: these can be Boolean, Binary, String, Text, numeric, etc.

- Codes and identifiers: there are two main code types in this version, the first set of codes are defined by HL7 and are used to structural attributes, the second type of codes are defined by other entities, like SNOMED [9]. An identifier is a code generated automatically used to provide a unique identity to people, organizations, objects, etc.
- Date/time
- Name and address: these data types are similar to the ones used in v.2.x

The organization has also developed a notation model called Refined Message Information Model (RMIM) that displays the structure of the message. This model can be presented as a color-coded diagram.

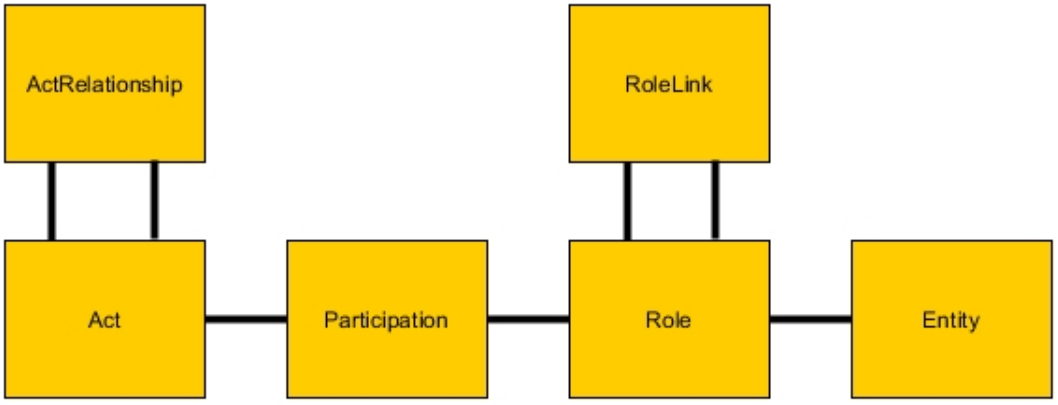

FIGURE 2.2: Reference Information Model (RIM)

2.3 IEEE11073

The ISO/IEEE 11073 [10] also known as x73 or is a family of standards developed by the Institute of Electrical and Electronics Engineers (IEEE) and the European Committee for Standardization (CEN) that were grouped by the International organization for standardization (ISO).The first part of this standard began being developed by IEEE in 1984 under the name of MIB. Another important part of the standard was independently developed by CEN in 1994 by the name of (PoC-MDC). This was already a set of standards to interconnect and interchange MD data. This standard was revised in 1999. Later in 2000/2001 both ISO and IEEE reached a consensus and created a joint project to create an international standard. To reach international consensus, CEN was also invited to the project.

This family of standards aims to respond to the interoperability issues of between-vendor MDs in the ICU by using the Poc-MDC standards for MD connection with the information system and a set of documents that specify each of the 7 OSI layers of a communication system.

The Poc-MDC standard solves the interoperability issues in the level of the MD by connecting all the devices to be monitored to a central gateway responsible of managing all communication between the MDs, retrieving data from the MDs and communicating that data to the EHR server. In some solutions [11] the MDs with proprietary communication protocols are connected to an adaptor responsible of converting the acquired data to the X76 data representation model and dealing with the communications with the gateway creating solutions that are end-to-end standards.

The first part of the communication standard worthy of mention is based on standards developed by CEN and specify basis upon which the rest of the standard is configured. It defines the nomenclature, syntax and semantics of the data, the Domain Information Model (DIM) and provides several specialization documents regarding MD categories.

- Domain Information Model (DIM)

The DIM is an object oriented model that represents real world entities like MD or patients in an abstract manner. This model is made up of eight packages:

The Medical Package deals with acquiring biomedical signals and correctly representing them, while also gathering context information necessary for the interpretation of those measurements. In 2.3 a general scheme for the Model of the Medical package is presented. Here the VMO (Virtual Medical Object) object is the base class of all the objects of this model. The VMD (Virtual Medical Device) in an abstraction of a medical device and contains objects representing measurements and status information. The channel object is used to group metric objects that can be of several kinds (Numeric, Real time, etc.)

The Alert Package contains objects to deal with information related to physiological alarms, techniques and information for the user about the equipment.

The System Package deals with the representation of devices that acquire or process vital signals information.

The Control Package contains objects that specify the data accessible by an remote system. It also implements means to control several parameters of the data, for example the sample rate.

The Extended Services Package contains objects providing management services for medical objects.

The Comunication Package contains the objects responsible for storing information regarding to how the devices communicate

The Archive Package supports objects related to the storage of the acquired signals

The Patient Package contains objects regarding the patient that are relevant to the device.

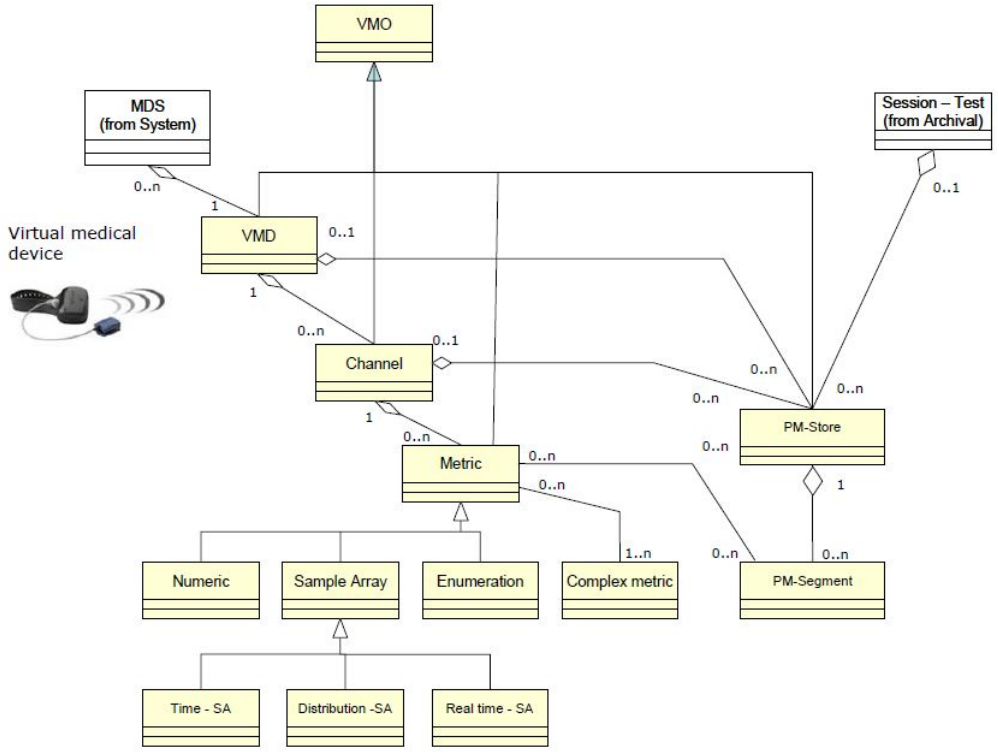

FIGURE 2.3: General scheme of the Model for the Medical Package

- Common Nomenclature

This part of the X76 protocol also defines document with a Medical Device Data Language (MDDL) based on the DIM. Here all the semantics and syntaxes used in the messages are specified and the all the nomenclature used is defined using 16-bit unique codes. This language contains thousands of terms regarding information about patients, MD, measurement values and methods, techniques, alarms, etc.

- Medical Device Application Profiles

The second part of the standard defines the application layer for exchanging data coded in MDDL. In this section the standard defines the services that will be used to communicate information between MD, Gateways and servers in the three upper layers of the OSI model, the application, presentation and session layers. In 2.4 is presented the generic communication stack and the encapsulation scheme of this part of the standard.

In order to be efficient in the implementation, the headers that are added in each layer of the communication model have a fixed structure with optional elements. This made the communication stack more flexible and allows other message transmission profiles to be accommodated in the structure. For this five protocols are defined:

The Common Medical Device Information Service Element (**CMDISE**) is responsible for providing services of data objects management like data retrieval or object creation.

The Association Control Service Elements (**ACSE**) provides mechanisms to establish or brake logical links between two applications of different systems.

The Remote Operation Service Element (**ROSE**) defines the means to invoke operations in a remote system by using request/answer elements.

The **Presentation Layer** responsible to negotiate the abstract and reference syntax

The **Session Layer** to provide support the ACSE standard

The Medical Device Information Base (MDIB) is specified MIB part of the standard and contains the names of the object instances of the DIM. In the MIB also defines two communication profiles, a Baseline Profile that mainly receives information via report events when there are changes in data or new data is available and a Pooling profile that allows the host to explicitly request data from the device.

The Standard also has standards for the remaining layers, for an overview of the x73 documents we refer to [12].

This standard is still under development but it is already possible to create end-to-end solutions by using it [11], several MD manufactures have also adopted the standard and implemented versions of it in their devices, one example is Philips. X73 is also being adapted to telehealth systems and has been adopted by the Continua Health Alliance [13] as its base standard.

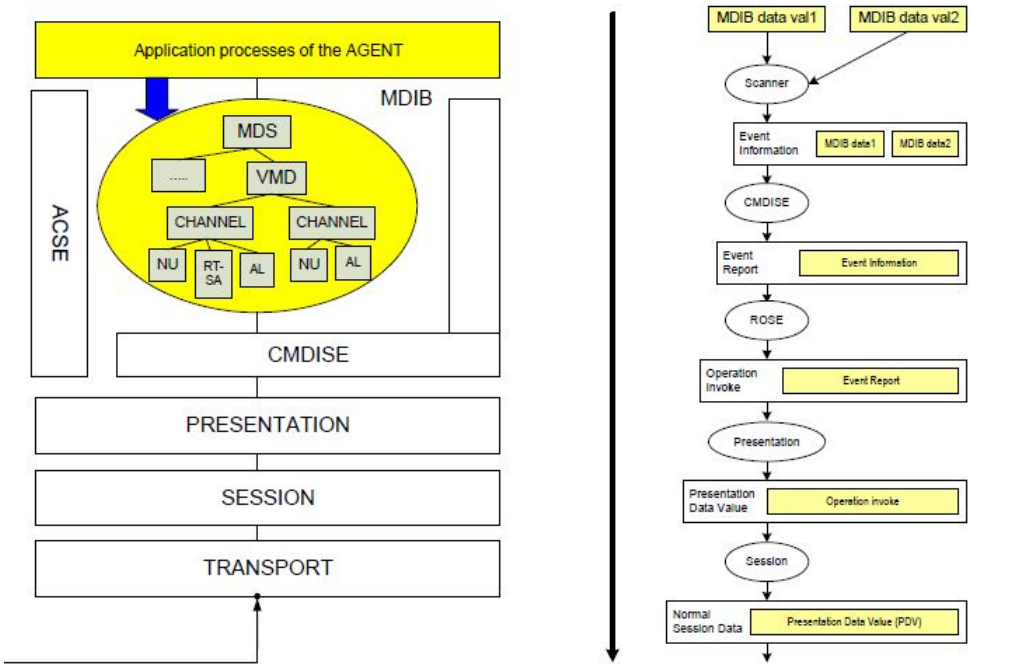

FIGURE 2.4: MDLL Protocols - Generic Communication Stack (left) Encapsulation Scheme (right)

2.4 Communication Protocols of the Integrated Medical Devices

This solution envisions the integration with all MD of the HSJ neurological patient's ICU. This unit is equipped with ventilators from two different vendors, the Maquet servo-I and the Draeger Evita. It also has infusion pumps manufactured by Fresenius and a BIS monitor manufactured by Aspect Medical Systems.

The communication protocols are different for each vendor and therefore four different drivers had to be implemented. In appendix D each of these protocols are briefly presented.

2.5 Similar Solutions on the Market

The application developed in this work intends to be an upgrade for the B-ICU.Care solution in the areas of MD integration and MD data sharing. With this new addition B-Simple intends to solve a particular problem in the HSJ neurological patient's ICU while adding more value to its flagship solution B-ICU.Care.

This solution is inserted in the Portuguese and Spanish market of ICU and anaesthesia information systems. In these markets the major competitors are the intensive care solutions distributed by iMD-Soft [14] and Otpum[15], formally known as Praxis. Both these companies offer a solution in everything similar to the one provided by B-Simple but no specific information regarding the nature of medical data acquired is provided the on these companies websites.

Still in the Portuguese market another solution worthy of notice is Alerts Patient Data Monitoring System [16]. This solution is also capable of at least acquiring medical data from MD, but this company is still to enter in the ICU information System market.

Capsule is one company responsible for the distribution of one of the leading solutions for ICU information systems [17] outside the Portuguese market. This solution is specialized in the integration of MD data in the Electronic Medical Record and provides a standard based broker for MD data sharing. Another company that also provides a solution specialized in the MD integration is distributed by True Process [18]. This solution is also capable of sharing MD data through an broker.

The addition of this new feature to B-Simple's solution will therefore provide competitive advantages regarding the Iberian market and will imbue the solution with more tool to enter new markets.

Chapter 3

Specification Analysis

In this chapter are presented the first stages of the software development, the requirement analysis, the system architecture and the software test protocol.

3.1 Requirement Analysis

This analysis was performed in the first 8 weeks of the development process. During this time all the necessary requirements were listed by conducting several meetings with all the major stakeholders. Due to the geographical distance between all the stakeholders it was not possible to conduct a meeting that gathered everyone. All the requisites were therefore gathered by using the three following methods:

- Informal week-to-week electronic meetings with b-simple's engineer Ricardo Sal,
- A face-to-face meeting with Ricardo Sal and Dr. Celeste Dias in the HSJ,
- E-mail conversations with Dr. Peter Smielewski, developer of ICM+.

From these meetings a Requirement Analysis document was produced and is available in the Appendix A of this thesis. In this document all the requirements are subdivided in four groups:

- Functional Requirements
- Nonfunctional Requirements
- Hardware and Software Requirements
- Technological and Architectural Requirements

The main functional requirements are related to the acquisition of data from medical devices, whether it is numerical or curve data (Requirements 1-6). Another important functional requirement was the presence of an HL7 broker capable of allowing external applications to select the data they need (Requirements 7-8).

The main non-functional requirements listed in the document are closely related to the need of constantly upgrading the application, whether it is by increasing the number of MDs supported or the types of query that the broker can receive (Requirements 11-13, 22). Another area covered is scalability, here requirements related to the capacity of the solution to perform well in any condition or volume of work and be capable of quickly restoring its function if anything goes wrong (Requirements 14-20).

In the Hardware and Software Requirements section are listed all pieces of hardware and software needed to perform the work.

Lastly in the Technological and Architectural Requirements is stated the programming language that the solution will be developed on and some important features of how the data acquisition process should have, namely the capacity of connecting with a MD using a serial connection or a TCP/IP one (Requirement 27).

Some of the requirements became partially or completely obsolete during the development process when the need of integrating with a database ceased. These requirements are the requirement 9, 13, 14,17 and 20.

3.2 System Architecture

3.2.1 Introduction

The second step of software specification is the system architecture. In this chapter an overview of the proposed architecture is presented. This architecture aims to produce a model capable of effectively responding to the listed requirements. In particular, with this architecture we aim to address the requirements of:

- Automatic detection of any supported MD
- Homogenous data representation
- HL7 Broker
- Extensibility
- Event logging

During the course of this work the initial architecture took on some changes. The most important of these changes will be mentioned when pertinent throughout the chapter.

3.2.2 Global Overview

With the analysis of the problem statement and the requirement analysis sections it is easily noticeable the existence of two major interfaces that the solution to be developed must interact with. The first interface, the MD Interface, is responsible of the communication with all connected MDs operating in the Unit. All these MDs and the solution under development are physically connected by a serial terminal server. This interface poses several development challenges:

- Each different medical device will usually communicate via a proprietary protocol. This means that the solution will have to support as many protocols as needed. It also means that the running solution will have to be able to detect in which of the supported protocols the MD is communicating.
- Most MDs are passive devices, meaning that they will not proactively try to initialise communication. To tackle this problem the solution will have to periodically probe all the ports where new MDs can eventually be connected.

The second interface is the External Communication Gateway and will be responsible for managing all the sharing of data from the medical devices with external applications. In this interface all intervening applications will always communicate using the HL7 protocol. The main challenge in this interface will be to devise a method to route the right data to each application communicating with the HL7 broker.

At the beginning of the project two more interfaces were also envisioned. These were the Database and the email service interfaces. The Database interface was removed from the project because it was decided that this solution was to work independently from the B-Simple solution and all communication would be made via the HL7 Broker making this way this interface obsolete. The email service interface was also removed from the solution and left to be done as future work.

In figure 3.1 is presented a diagram depicting the general overview of the system.

3.2.3 Application Modules

The developed application was divided in two main modules, the Data Acquisition and the External Application Communication modules. With each module incorporating all

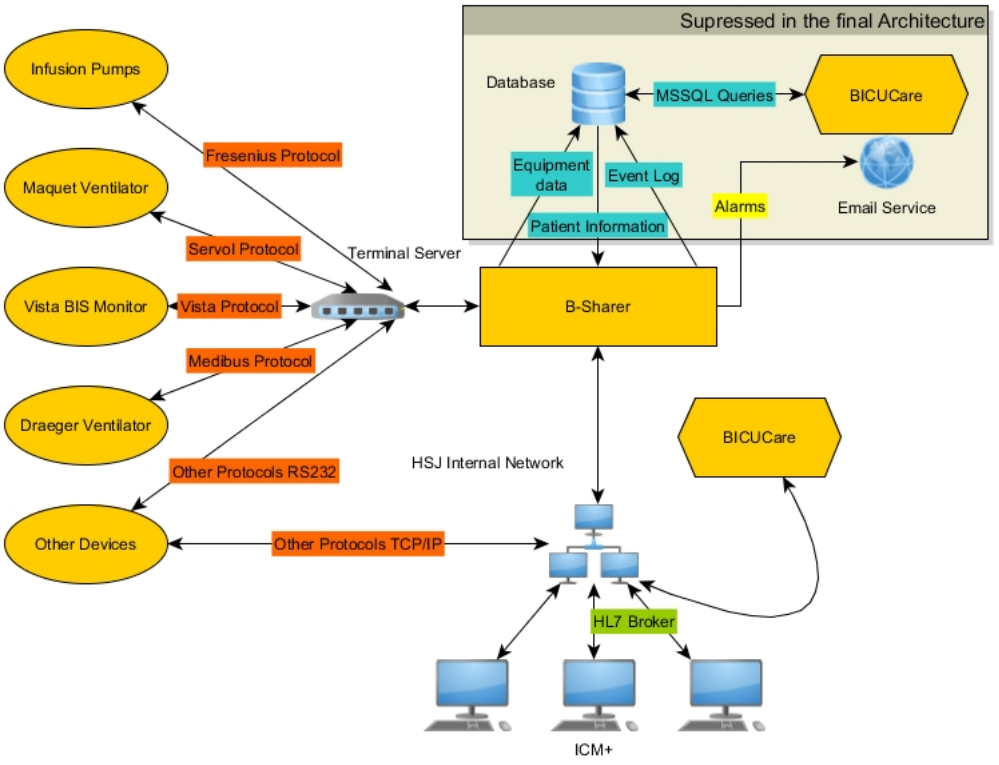

FIGURE 3.1: Global Overview of the System

the necessary tools to properly handle one of the interfaces stated above. The main responsibilities of these modules are:

- **Data Acquisition module** : managing all the communication and data acquisition processes with the medical devices and generating data objects to be used by the other modules of the application.
- **External Application Communication module**: managing all the communication with external applications requiring data from the MDs. This module will use the data objects generated by the Data Acquisition module as the source of MD data.

Alongside these two modules will be running another called Event Logger that will be responsible of logging all the important information regarding the normal functioning of the application. As a future work this module will also generate emails containing exceptionally important information that would require immediate attention from the support team. In figure 3.2 and overview of the application modules is presented.

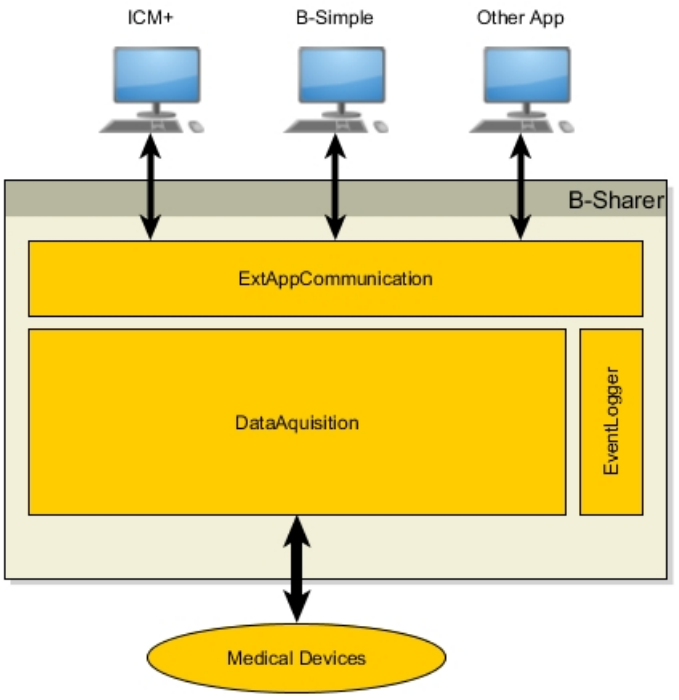

FIGURE 3.2: Application Modules

3.2.4 Data acquisition Module

This module is divided in four main sections:

- The Data Acquisition Manager responsible of orchestrating the creation of probing and data sessions, and managing all the data generated in the module.
- The Probing Session is the entity responsible of probing a port in order to detect a MD and the protocol he communicates with.
- The Data Session is the entity responsible of orchestrating the communication with a given medical device and generating data objects containing data from that MD.
- The fourth section of this module is the MD Driver Library that is composed of a collection of DLL files that are used by both the probing and data sessions.

The diagram depicting this module is presented in figure 3.3.

3.2.4.1 Data Acquisition Manager

In order to be able to orchestrate the creation of Probing and Data sessions the Data Acquisition manager will keep track of the state of every port potentially being used by

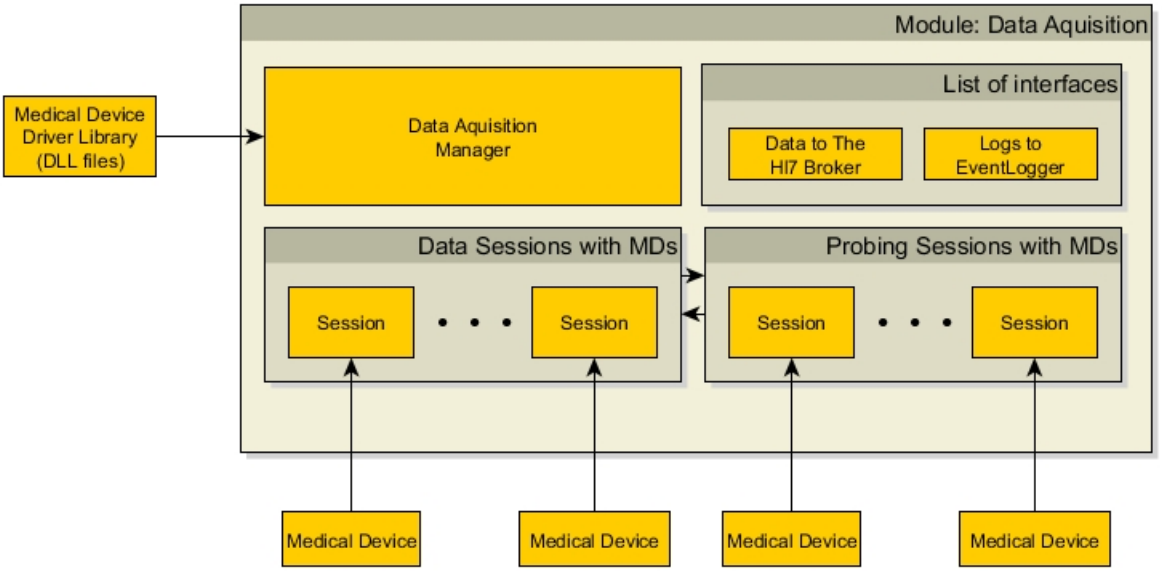

FIGURE 3.3: Module Data Aquisition

the MDs. It will then periodically create probing sessions on idle ports. If any probing session would find a supported MD it would inform the Data Acquisition Manager of the occurrence and a new Data session will be created in that port. This manager will also have to maintain a buffer to accommodate all the data objects generated by the data sessions and providing an interface to the External Application module so that module can have access to these data objects. This buffer will also have to be managed in order to keep the amount of data in the buffer workable and relevant. In figure 3.4 a diagram depicting the Data Acquisition Manager is presented.

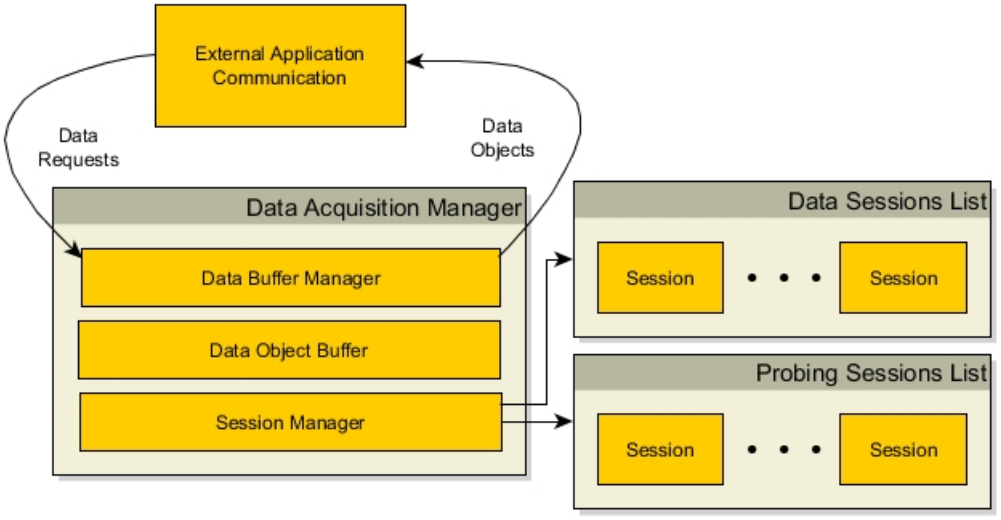

FIGURE 3.4: Data Acquisition Manager

3.2.4.2 MD Driver Library

The MD drivers in this library will be used by both the Probing and the Data sessions. These drivers are responsible of processing the messages issued from the MDs and creating the messages to be sent to the MDs. These drivers will also provide the Data Session with data so that it can generate data objects. In figure 3.5 a diagram depicting a MD driver is presented. The architectural objects presented are:

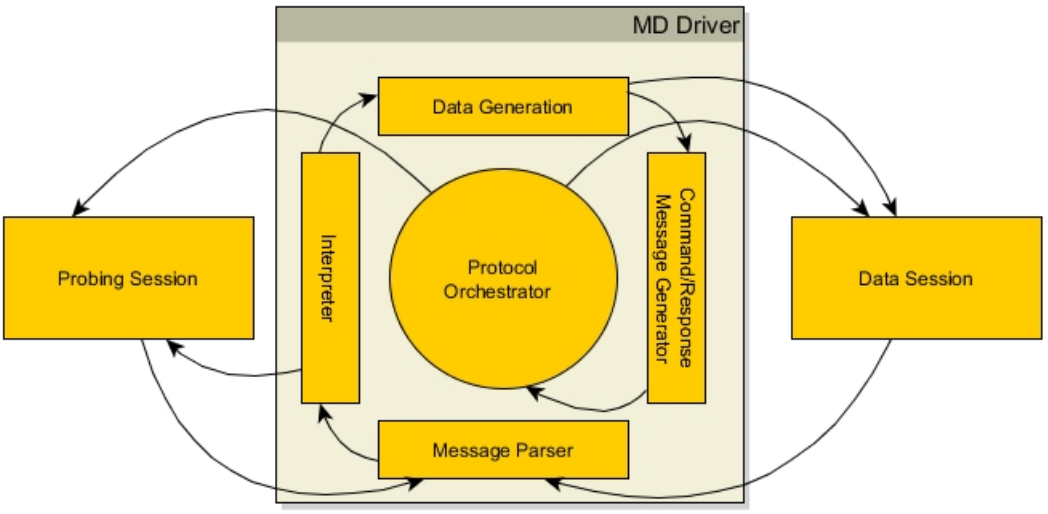

FIGURE 3.5: Supported Equipment Drivers

- The Protocol Orchestrator is responsible maintaining the communication with the MD by managing the timings of message generation and keeping track of the state of the protocol,
- The Message Parser will use the byte stream received from the Session that is implementing it to generate valid messages,
- The Interpreter will be responsible of extracting the information contained in a received message. If a probing session is implementing the driver, the interpreter will be also used to identify the MD in question.
- The Data Generation will be responsible of extracting all the medical data from the messages and issue it to the Data session implementing it.
- The Command/Response Message Generator will be responsible of creating all the messages that need to be sent back to the MD by the Protocol Orchestrator.

3.2.4.3 Probing Session

The probing session will establish a connection between the application and the MD. In order to detect the MD present in the port it will iteratively try to initiate communication by using one of the protocols implemented by the MD drivers. When a MD starts responding to the initialization commands the Probing Session will inform the Data Acquisition Manager of the occurrence and terminate. In figure 3.6 a diagram depicting this Session is presented.

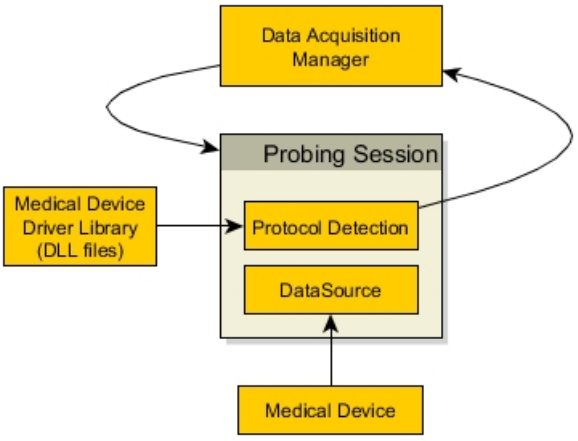

FIGURE 3.6: Probing Sessions

3.2.4.4 Data Session

When a MD is detected a Data session will be created. This session will also establish a connectionr between the application and the MD. It will also gather all the data generated by the MD and create Data Objects. These objects will then be sent to the central buffer in the Data Acquisition Manager. In figure 3.7 is presented a diagram depicting the Data session.

3.2.5 External Application Communication

The diagram depicting this module is presented in figure 3.8. This module is divided in three main sections: The server, the Subscription Manager and the Subscriber.

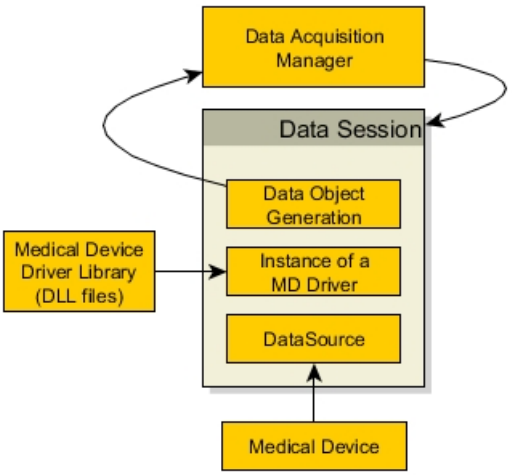

FIGURE 3.7: Data Session

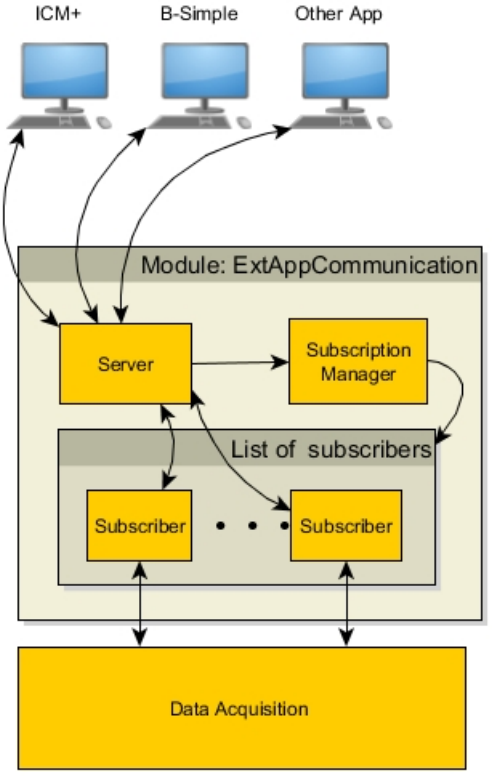

FIGURE 3.8: Module External App Communication

3.2.5.1 Server

The Server is an architectural object responsible of maintaining a server capable of sending and receiving messages from external applications.

3.2.5.2 Subscription Manager

The Subscription Manager will manage the creation of subscribers.

3.2.5.3 Subscriber

The Subscriber unit will be responsible of managing all communications with a single application. The diagram of this object is presented in Figure 3.9. It implements 5 architectural objects:

- The Query Parser is responsible of gathering all relevant information from the query.
- The Subscription Manager will use information from the parser in order to configure the Data Object Retrieval, retrieve messages from the buffer and handle the communication with the external application.
- The Data Object Retrieval will be used to gather all the relevant data object from the Data Acquisition Module.
- The Message Buffer will store all the messages that are in queue to be sent to the application.
- The Message Creator will transform Data Objects in messages.

3.2.5.4 Event Logger

This module will receive information regarding events that occurred in the other modules of the application and stores it in log files. This part of the application was provided by B-Simple.

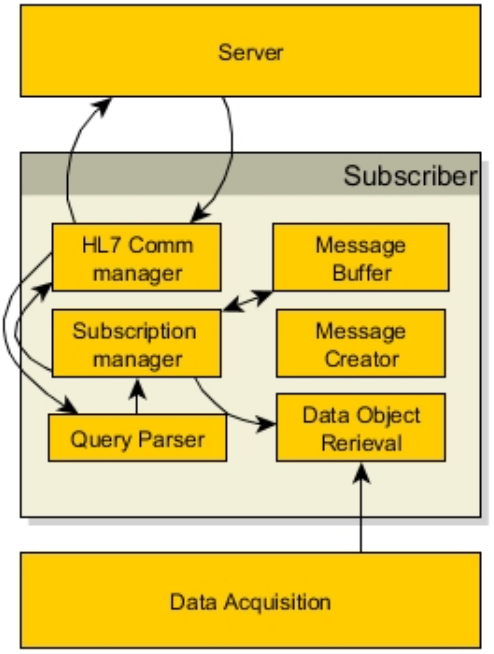

FIGURE 3.9: Subscriber

Chapter 4

System Implementation

In this chapter an overview of the implementation phase will be presented. As stated in the Requirement analysis the solution was developed in VB.net. In this phase all the described architectural objects were implemented, from this a fully functional prototype was developed.

In order to aid in the development process, B-Simple has provided a test tool that was capable of issuing both MD and HL7 messages. It has also provided the Event Logger class, a base for the Hl7 broker and a class that parsed Hl7 messages and provided methods to retrieve information contained in the message. It also provided a class that was capable of maintaining a TCP socket.

4.1 Implementation phases

As stated in the Software Development Methodology section the implementation phase was conducted incrementally. The first architectural object implemented was a MD driver, and in order to produce a testable prototype, part of the data session was also implemented in order to handle the physical communication with the MD. This prototype passed through a preliminary test where it capacity to establish communication and retrieve medical data were tested.

After studying the results of the preliminary tests, a new iteration on the developed prototype was conducted in order to correct some bugs and the rest of the MD drivers were implemented. With the experience gathered during the previous phase, the implementation of the new drivers proved faster. As in the end of the previous phase, the prototype developed was tested and reiterated.

The third development phase produced, after its corresponding tests and reiterations, another functional prototype. This prototype contained a fully developed Data Acquisition Module containing all the Architectural objects implemented. This prototype was capable of generating data objects from MD.

The fourth increment produced a prototype that implemented the interface between the Data Acquisition module and the External Application Module. Here the part of the Subscriber that retrieves information from the Data acquisition and the part that created HL7 messages were also implemented. This prototype was capable of acquiring data from the devices and producing Hl7 messages with the medical data.

In the fifth and final increment the rest of the subscriber was implemented alongside with the HL7 broker and the Subscription manager. In this last increment, the HL7 message protocol was also created and implemented in the solution. The resulting prototype was a fully functional solution that was capable of both acquiring data from MD and maintaining an HL7 broker capable of sharing that data with other applications.

4.2 Configuration file

In order to make it easy to configure the developed solution to different medical units, an user interface was developed. Here it is possible to:

- Associate hospital beds with ports,
- Define the sleep time for all data related timers,
- Define the IPs and ports of the HL7 broker,
- Insert invariable information to be used in the Hl7 messages, like the hospital name.
- Configure the folder where the Dll files containing the MD drivers are to be placed in.

This user interface then generates an xml file that is parsed at the initialization of the application.

4.3 Data acquisition Module

As stated above, this module is responsible of acquiring all data from the medical devices and generating Data objects to be used by the External Application Module to create HL7 messages.

4.3.1 Data Acquisition Manager

This is the base class of the Data Acquisition Module and can be divided in two main sections, the Session Manager and the Data Buffer Manager.

The **Session Manager** is responsible for handling the creation and destruction of sessions. For this the session manager uses a Timer that periodically checks for idle ports and initializes new Probing Sessions in those ports. This timer makes use of an array containing the status of each port configured in the solution. Here a port can have one of three statuses, the Idle, the Probing and the Data Status.

The overall logic for the creation of session is as follows: after a Data session is terminated, the method **SessionCloser** is called and the status of the port is set to idle. The port will stay in the idle status for a configurable amount of time. After that time, the timer will create a new probing session on that port by using the method **ProbingSessionCreator**. If the probing session detects a new device the method **SessionCreator** is called so that a new Data Session is created in that port. If no MD is detected the method **OnClosingProbingSession** is called and the port is set to idle again.

In this module all sessions are self-destructed, therefore the Data Acquisition Manager will only receive events from the sessions informing its termination.

The Probing Sessions will raise the **CanStartSession** event, that calls the **SessionCreator** method, when a MD is detected and the **OnProbingSessionClosing**, that calls the method **OnClosingProbingSession**, when no medical device is detected.

The Data Sessions raise the **OnSessionClosing** event, that calls the method **SessionCloser**, when they terminate.

The methods called when sessions are closed are also responsible for eliminating the session objects from the Array List that is holding them. Therefore the methods used to create sessions are also responsible of adding these sessions to the corresponding Array List.

The **Data Buffer Manager** is composed by two independent timers and two data buffers. The timers will periodically check for old data in one of the data object buffers

and deletes will it if it is considered old. A Data Object is considered old after it is in the buffer for longer than a configurable amount of time.

These buffers are instantiated at the moment of the class construction. One of these buffers is responsible of storing numeric data objects and another for storing curve data objects.

This section also handles all the access to the Data Buffers by providing thread safe methods to insert and access data in the buffer. The **AddToBuffer** is the method used by the data sessions to populate the Numeric Data Object buffer. This method is called after the Data session raises the **OnDataObjCreated** event. To populate the Curve Data Object buffer, the method **AddToRTBuffer** is called after the session that created the data object raises the **OnRTDataObjCreated**.

The External Application module will access the data in the buffers by using three different methods:

- The **GetDataObjectList** method is used to retrieve all numeric Data Objecs that were inserted in the buffer after given date and corresponding to a given subset of ports.
- The **GetNewestDataObjectList** method is also used to retrieve numeric data objects, but here for each port only the newest data object will be retrieved.
- The **GetCurveObjectList** method is in all similar to the **GetDataObjectList** but here the buffer used is the one responsible of storing curve data.

The input variables for these methods are therefore an array of ports and a date corresponding to the last time the buffer was accessed by the subscriber calling it. All these methods and timers are controlled by two semaphores, one for each buffer that will only allow access to the buffers to one thread at a time.

4.3.2 Probing Session

This class is instantiated by the Data Acquisition Manager when a port is Idle for a configurable amount of time. The Probing Session begins by instantiating an object for every MD driver present in the DLL library using assemblies. It will then get a message of initiation of communication from each driver. These messages will then be periodically sent to the medical devices by a timer called **ProbingSessionTimer**.

This Session will also instantiate an object responsible for creating a TCP socket that will handle the physical connection with the MD. From this object two main events can be raised:

The first is called the **OnConnectionBroken** and is raised when the TCP connection goes down. This event calls a method with the same name that will try to re-establish the TCP connection.

The second event is the **OnReceiveData** that is raised when a steam of data from the MD is ready to be processed. This event will also call a method with the same name that will send a copy of the byte stream to each instance of MD drivers.

When a Probing Session detects a valid "start of communication response" from a MD it will raise an event called **OnProtocolFired** that will call the method **DetermineProtocol**. This method will determine the name of the protocol and raise the method **CanStartSession** to inform the Data Acquisition Manager that a MD with that communication protocol is connected in that port and that a Data Session can be initialized.

The method **DetermineProtocol** will also be responsible of terminating the Probing session class and releasing all objects from memory. If no MD is detected after a period of time, the method to close this session will be the Close method and the event raised to inform the Data Acquisition will be the **OnProbingSessionClosing**.

4.3.3 Data Session

This class is also instantiated by the Data Acquisition Manager object, but here the method that instantiates it is the **SessionCreator**.

Similarly to the Probing session, this class also will instantiate an object responsible for creating a TCP socket where the same events are raised.

In contrast to the probing session, here only one MD driver is instantiated. The MD drivers will raise four different events:

- The **OnSendMessage** event is raised when a message needs to be sent to the MD. This event calls the method **SendMessage** that will queue a message to be sent by the socket to the MD.
- The **OnProtocolClosing** event is raised when the Driver detects that no more messages are being sent to him, in this case it will self-destruct and warn the Data Session of it. The session will then call the method **closeProtocol** that will terminate the session and warn the Data Acquisition Manager of the occurrence.

- **OnDataGenerated** event is raised when the Driver finishes parsing a set of MD messages that contain Numeric Data. The session then calls the method **CreateDataObject** that will create a Data Object and raise the event **OnDataObjCreated** in the Data Acquisition Class so it can be added to the numeric data buffer.
- **OnRTDataGenerated** event is raised when the Driver finishes parsing a MD message that contain Curve Data. The session then calls the method **CreateRTDataObject** that will create a Curve Data Object and raise the event **OnRTDataObjCreated** in the Data Acquisition Class so it can be added to the curve data buffer.

4.3.4 MD Driver Library

This library is composed by a set of DLL files containing the drivers that can process and create messages to be sent to the medical devices. These drivers are instantiated by both the Probing and Data sessions.

Even though all communication protocols with MD are different, these files were constructed so that the interface with the sessions is consistent through all drivers.

The interface with the sessions is conducted through the five events previously mentioned earlier:

- **OnSendMessage**
- **OnProtocolClosing**
- **OnProtocolFired**
- **OnDataGenerated**
- **OnRealTimeDataGenerated**

And through the method **ToMessage** that is called by the session that instantiated the MD driver when the **OnRecievedData** event is fired. This will redirect the received byte-stream to the MD. The **ToMessage** method is responsible of parsing the byte stream into valid messages from the protocol the MD driver implements.

When a message is generated by the **ToMessage** method, the **Interpreter** method is then called. This method will be responsible of extracting information from the messages and sending that information back to the Session implementing it. This information can

be the name of the MD if a probing session is instantiating it or data objects if it is under a Data Session.

The driver also provide a timer responsible of orchestrating the cadence of messages to be sent to the MD in order to achieve a steady flow of data. This timer will also be responsible of terminating the communication with the MD and triggering the termination of the Sessions when the MD stops receiving data from the MD.

4.3.5 Data Object

This is the generic object used to transport numeric data throughout the application. This object is instantiated by the Data Sessions when data related events are fired and is used by the External Application Module to create Hl7messages. It is composed by several variables:

- The **device** variable will contain the name of the protocol, and therefore the device that originated the data
- The **timestamp** variable will be used to contain the timestamp of the moment of the generation of data coded as and Hl7 date type.
- The **internalTimeStamp** variable will be used to contain the timestamp of the moment of the generation of data coded as the processor time. This variable will then be used in all timing calculations, whereas the previous time stamp will be used to create the HL7 messages.
- The **ip** will contain IP address of the Terminal served that the device is connected to.
- The **port** will contain the port number of the Terminal Server that the device is connected to
- The **numericData** variable will contain a list of string arrays, with each array containing the value of a data element, the generic code used by the application to represent the data element, the units in which the element is encodded in and an abbreviation of the common name given to the data element.

4.3.6 RT Data Object

This object is equal in everything to the Data Object presented above, but in the **RT-data** variable that substitutes the **numericData**. This object will be used to transport Curve data through the application.

The **RTdata** variable in this object will also be composed by a list of string arrays the will contain a string representing the array of data containing the curve, the generic code used by the application to represent the data element, the units in which the element is encodded in, an abbreviation of the common name given to the data element and the sampling rate at which the data was generated by the MD.

4.4 External Application Communication

This module is the one responsible of maintaining an HL7 broker by using the data objects created by the Data Acquisition Module. This module is divided in three main sections, the Subscription manager, the Hl7 Broker and the Subscriber.

4.4.1 Subscription Manager

This class implements a TCP server that is capable of receiving connections from other applications through a listener that creates a new TCP client when a new connection is received.

This listener is associated to a thread that periodically checks when a new client is created. When this happens the subscription manager will compare the IP of the new client with the IP of all subscribers running in the application. If there is no subscriber with the IP, a new subscriber is created. If there is already a subscriber communicating with that IP, the manager will use a method of that subscriber called **Reconnect** in order to re-establish the connection with that application.

When a subscriber terminates the communication definitively, an event called **OnSubscriberClosed** is raised. This calls the method **CloseSubscriber** that will remove the subscriber from the subscriber list and make that IP available again.

4.4.2 Subscriber

The subscriber incorporates all the necessary methods to interpret Hl7 queries, create Hl7 messages with data from MD and sending Hl7 messages.

In order to handle the socket and the HL7 communication protocol, and object of the class **HL7Protocol** is instantiated.

When the HL7protocol object receives a message, it creates an **HL7Data** object with it and raises an event called **OnHl7ParseEnd**. When this event is raised the Subscriber

will check if that message is of the HL7 QRYˆR02 type and proceeds to process it in order to update the data subscription. If the received message is of the ACK or NAK type, a different event is raised. The **OnHL7ACK** also retrieves an **HL7Data** object and is used by the subscriber to check for its corresponding message in the message buffer in order to delete it.

After a subscription is made, a timer is instantiated that will retrieve the subscribed data objects from the Data acquisition module using the **GetDataObjects**method. This method will retrieve information using one of the three methods previously mentioned:

- The **GetDataObjectList** method,
- The **GetNewestDataObjectList** method,
- The **GetCurveObjectList** method.

Whenever a new query is received all aspects of the subscription will be updated.

If the **GetDataObjects** method retrieves numeric data objects an HL7 message is created by the **CreateHL7** method and sent to the client. If the data object received is relative to a curve the **CreateRTHL7** method is called instead and an HL7 message containing curve data is generated.

These messages are also inserted in a Message buffer that will keep it in memory until the corresponding ACK is received. The messages in the buffer have a configurable life-span and are periodically resent if they remain in the buffer. The Subscriber will continue generating and sending new messages to the subscriber, but it will time-out and close if no messages are received for a period of time.

In the HL7 Integration Protocol chapter a description of the messages exchanged and overall communication flow can be found.

Chapter 5

Hl7 Integration protocol

The main objective of this work is to produce data sharing tool that can be easily integrated with. In order to do this, a set of communication rules were created having the version 2.4 of the HL7 protocol as a base.

When using this protocol the external application must act as the master and take initiative when initializing communication. This is done by creating a TCP client in the server provided by the developed application.

After having the communication established, the external application will have to subscribe data from medical devices. This is done by issuing query messages of the type QRYˆR02 to the developed application. These queries can configure several aspects of the subscription:

- Add or remove medical data from beds operating in that unit from the subscription,
- The cadence a which messages are generated and sent,
- The mode of message creation; The external application can retrieve one message with data from that moment, or a message with all the data generated in the subscribed beds after the last message was sent,
- Subscribe numeric data, curve data or both,
- Completely unsubscribe and close connection.

In this protocol subsequent query messages will update the subscription. Any queries received by the broker will generate an ORFˆR04 informing the external application of the status of the subscription.

After the data is subscribed, the broker will periodically generate ORUˆR01 messages with the medical data. If more than one bed is subscribed, the broker will generate a different message for each bed.

All ORUˆR01 messages must be responded to with an HL7 ACK message, otherwise they will be periodically resent to the external application.

For a more detailed explanation on the actual structure of the messages exchanged, the supported MD and the data codes used to route the data elements, consult appendix C.

Chapter 6

Test Phases

During the development of the application five preliminary and one final testing phases were conducted. Each of the five preliminary testing phases was intended to test the new features of one of the functional prototypes that resulted from each of the development phases. The testing scenarios for each of these testing phases were:

- The testing scenario for the **First testing phase** was the running prototype and one instance of the testing tool provided by B-simple, simulating the Draeger ventilator. This test intended to access the capability of the solution to communicate with the medical device and generate a stable flow of data from the equipment.
- In the **Second testing phase** two scenarios were envisioned. The first testing scenario was equal to the one conducted in the first testing phase and intended to test the same features, but now for each of the medical devices that the solution had to integrate with. The second testing scenario was conducted alongside the real equipments. This scenario intended to prove that the drivers would also work with real devices.
- The **Third testing phase** was again performed by resorting to the testing tool. Here several instances of the tool were acting as medical devices. The main purpose of this testing phase was to test the MD device detection algorithm, the creation of multiple data Sessions and the generation of data objects.
- In the **Fourth testing phase** the testing scenario was the same as in the third phase, but here the aim of the test was access the prototype's capabilities of generating HL7 messages by using data objects.
- The **Fifth testing phase** already used a prototype that implemented all the features implemented. The testing scenario here was similar to the one of the two

previous testing phases, but here three new instances of the testing tool were also present. These three new instances were intended to simulate external applications trying to access data from the HL7 broker. Two of these instances were running in the same computer and the third was running in another computer present in the same internal network.

The main intent of this testing phase was to test the creation and life-cycle of subscribers and that each subscriber received the correct HL7 messages.

After these preliminary testing phases, a final one was conducted. In this testing phase the MD drivers were again tested with real equipments in order to revalidate the stability of the drivers and to validate the integrity of the data present in the data stream through the application.

The solution was then tested in the central server of the ICU of Braga's hospital. In this scenario, the solution was responsible of acquiring all the data from the MD of that unit. The solution was also connected to the B-Simple solution through the HL7 broker.

This scenario was intended to validate the capacity of the solution to keep subscribers resulting from real applications and to test the capacity of the solution to keep a stable run through long periods of time in the real world.

During this last testing scenario, it was not possible to test Real-Time related events, because in that unit, no supported BIS monitor was present. Therefore these features were only tested in the simulated environments.

On Appendix B is present more information regarding the actual tests performed during each of the testing phases.

6.1 Final Test Results

The final testing phase proved that the solution answered positively to 36 of the 41 tests performed. From these 5 remaining tests, the answerer to 3 is also positive, but some problems worth mentioning were detected:

- One of these tests envisioned a continuous operation of the developed prototype during at least 48 hours in a real environment. The solution was capable of handling all the tasks presented to it smoothly and without any errors. During this time the CPU usage never surpassed 5% and the RAM was always between 30 Mb and 40 Mb. The solution exhibited however a small tendency to increase the

RAM usage during the 48 hours. This increase was not higher than 5 Mb. During this test the solution also appeared to accumulate handles.

This problem was immediately addressed and the inappropriate release of some handles was found. The solution was again tested in the simulated environment and the problem was apparently solved. However scheduling a new 48 hour test was not possible and because of this, the solution could not transition to production.

- The other test was the CPU usage. As mentioned above the solution when tested in a real environment did not use more than 5% of the CPU, but here the driver of the BIS monitor was never used because Braga's ICU did not use this equipment. In the tests with the BIS monitor CPU usages of more than 20% were verified. This problem was extensively studied and the origin was found to not be related to the developed driver or to the application. It seems to be related to the way data is acquired form the serial port, because the difference in the CPU usage in a scenario where the solution is just acquiring data from the port and a scenario where the solution is acquiring data and generating HL7 messages is hardly noticeable.

  The logic used to acquire data from the serial ports is the same for all drivers, but this is the only driver where this happens.

- The remaining test was the RAM usage test. The problem here was in everything similar to the CPU usage problem, but here the RAM used increased when acquiring data from this protocol.

These last two problems will prevent the usage of the driver relative to the BIS monitors in a real environment until they are solved. In the test where these problems were verified the solution made use of a RS232 to USB converter in order to acquire data from the serial port whereas in the real environment data is acquired via a Serial terminal and the data reaches the solution via a TCP connection. The problem can be related to this fact, but this possibility was not tested because the serial terminal in HSJ was not accessible.

The last two problems where the solution answered negatively were the capability of restoring operation when the system crashed, and the capability of blocking applications that continuously issued corrupt messages. This happened because both these functionalities were not implemented. The restoration of operation will only be implemented when the solution transitions to production as it will be transformed in a window service.

Chapter 7

Conclusion

The developed solution provided a suitable answer to the most important requirements, in particular the developed solution encompasses:

- The capacity of dynamically detecting and acquiring data from MD,
- A functional HL7 broker capable of sharing medical data with any application present in a ICU,
- Easiness in the integration with new MD and in improving the complexity of the queries supported due to the modular philosophy employed in the development of the solution,
- Capacity of keeping data backups when the communication links become disabled,
- A solution easily configurable, allowing it to be installed in different medical units with minimal effort.
- The capacity of handling information from all MD and in an ICU unit and of distributing that information through all the applications in the unit.
- A solution where faults are handled in a way that allows most of the crashes to be prevented and where all interface situations are properly threaded so that the occurrence of blocks is minimized.

The developed solution did not envision however any extensibility regarding the use of other communication protocols in the broker. This happened because the decision of implementing the solution's broker using the HL7 standard proved to be enough, due to its versatility and widespread use. By using this standard, the solution became capable of integrating with the majority of the applications present in a medical unit.

The testing phase proved that the developed solution will be ready to transition to production and start managing the acquisition of data in an ICU unit when a final 48 hour test in a real environment is repeated in order to ensure that the remaining bug was effectively removed.

Appendix A

Requirement Analisys Document

In order to list all the necessary requirements, several meetings with all the major stakeholders were conducted:

- Informal week-to-week electronic meetings with b-simple's engineer Ricardo Sal,
- A face-to-face meeting with Ricardo Sal and Dr. Celeste Dias in the HSJ,
- E-mail conversations with Dr. Peter Smielewski, developer of ICM+.

In this document all the requirements listed were subdivided in four groups:

- Functional Requirements
- Nonfunctional Requirements
- Hardware and Software Requirements
- Technological and Architectural Requirements

A.1 Functional Requirements

Table A.1: Functional Requirements

Req.	Functionality	Requirement description
Req.1	Ventilator Recognition	Detection of any supported ventilator that is connected to the network followed by the establishment of communication and data transfer.

Req.2	Infusion Pump Recognition	Detection of any supported Infusion pump that is connected to the network followed by the establishment of communication and data transfer.
Req.3	BIS Monitor Recognition	Detection of any supported BIS monitor that is connected to the network followed by the establishment of communication and data transfer.
Req.4	Homogenous data representation	The solution should be capable of generating a common representation for data with the same semantic value, but that was acquired with different formats. (ex. data from ventilators from two different manufacturers).
Req.5	Acquisition of data relative to curves	The solution should only be able to acquire curve data form the devices when that data is specially required.
Req.6	Acquisition of numeric data	The solution should be able to acquire all numerical data available on the devices at a configurable frequency.
Req.7	HL7 query processing	The solution should be able to process incoming HL7 messages from the 2.4 version and extract valid information from it.
Req.8	HL7 message generation	It should be capable of generating HL7 (v.2.4) messages containing: Numeric Data and Curve data.

A.2 Nonfunctional Requirements

Table A.2: Nonfunctional Requirements

Requ	Functionality	Requirement description
Req.09	Database inaccessible to other applications	The solution should guarantee that no other application can insert or access information present in the database.

Req.10	Patient data should be masked	The solution should guarantee that in the HL7 message, the data from patients should be masked.
Req.11	Medical equipment extensibility	The solution should be developed in a way that ensures that future additions of medical device interfaces are easily integrated.
Req.12	Extensibility on the inter application communication gateway	The solution should be developed in a way that ensures that future additions of inter application interfaces are easily integrated, whether they are new types of HL7 queries or other communication protocols.
Req.13	Extensibility on the database communication	The solution should be developed in a way that ensures that future additions of other database interfaces are easily integrated.
Req.14	Communication Recovery	The solution should be able to recover the communication links with any medical device or the database if the communication is not closed correctly.
Req.15	Message Validation	The system should verify the validity of every message received, whether this message is derived from a medical device or an external application.
Req.16	System robustness	The solution should be capable of recovering from any fatal error that might occur. It should also be able to recover from non-fatal errors without compromising the rest of the solution.
Req.17	Data backups	If the connection with the database is lost, the system should be able to generate a data backup and send it back when the communication is re-established.

Req.18	Scalability	The solution should support the addition of any numbers of medical devices or external applications without compromising its performance. The solution should also guarantee that the resource consumption does not compromise the performance of other services running on the same machine.
Req.19	Continuous availability to receive messages	The solution should be able to receive all queries sent from external applications at any given moment.
Req.20	Event Logger	The solution should be able to keep a log of all the relevant events related to medical devices, database communication, external applications and internal events.
Req.21	Critical event alarm system	The solution should be capable of warning the support team and authorized personnel of any critical event occurring in the running solution via an email service.
Req.22	Reconfigurable system	The solution should be developed in a way that it is easily installed and configures in a different hospital.

A.3 Hardware and Software Requirements

Table A.3: Hardware and Software Requirements

Req.	Functionality	Requirement description
Req.23	Server	It will be necessary an windows machine to be used as a server with sufficient capacity to deal with all the data being acquired from the medical devices in the healthcare unit and all data being transferred to other applications running on the ICU.

Req.24	BIS monitors form the Aspect medical systems	The device should support numeric and curve data communication
Req.25	Orchestra infusion pumps	The device should support numeric data communication
Req.26	Draeger and Maquet ventilators	The device should support numeric and curve data communication

A.4 Technological and Architectural Requirements

Table A.4: Technological and Architectural Requirements

Req.	Functionality	Requirement description
Req.27	Communication with medical devices	The solution should communicate with medical devices by RS232 and by TCP/IP. All supported devices should be plug-and-play
Req.28	Database Integration	The developed solution should be capable of populating a database shared with the B-ICUCare resorting to MSSQL queries
Req.29	Simultaneous data acquisition from multiple devices	The solution should be capable of supporting the simultaneous data acquisition from all the medical devices running on the healthcare unit.
Req.30	Programming Language	The solution shall be implemented in VB.net

Appendix B

Software Test Document

B.1 Introduction and Objectives

In this document the test pile that will be used to evaluate the developed solution is presented. The tests described in this document aim to verify that all the requirements have been fulfilled and all the modules are performing correctly.

The test phase was itself divided in five preliminary sub-phases and one final phase where the solution was tested in the real world.

At each new test sub-phase the test performed in the previous phases were repeated and new tests were added to the test pile.

B.1.1 Preliminary Test 1

In the first testing phase the prototype was tested for its capability of connecting to a Draeger ventilator and establishing a stable communication with it by using the communication protocol specific to the MD.

For this the prototype had to establish a data link with a MD emulator provided by the B-Simple company.

B.1.2 Preliminary Test 2

In the second testing phase the process performed in the first phase was repeated, but here all drivers developed were tested with the emulator and with the actual MD.

B.1.3 Preliminary Test 3

In this section the prototype's capabilities of generating Data objects from data acquired form medical devices was tested. Here the developed logic for handling the creation and destruction of sessions was also tested.

The scenario in this phase involved five MD emulators, each representing one medical device. All these emulators were running in the same computer as the running prototype.

B.1.4 Preliminary Test 4

In this phase the ability of the prototype to generate HL7 messages by using the Data objects generated by the Data sessions was evaluated.

The scenario here was the same as the scenario of the third test phase.

B.1.5 Preliminary Test 5

In this phase the prototype already contained all features of the final solution. This prototype was inserted in a simulated environment where it was tested when handling communications with 5 emulated MDs and three emulated subscribers. Both the MDs and two of the subscribers where running on the same computer, the other one was in another computer connected to the same network.

Here the capabilities exchanging HL7 messages through the HL7 broker with multiple subscribers while handling multiple subscriptions was tested.

B.1.6 Final Test

This final test phase was conducted in two different places. The first place was Hospital S. João where the solution was tested with the actual MDs in order to further validate the final MD drivers.

The second place was the Hospital of Braga where the solution was run in the central server for two days. Here the solution was responsible of acquiring all the Data generated by connected MDs and serving a B-Simple Subscriber that received all the data generated.

B.1.7 Test Pile

The next section presents a table containing all performed tests. For each test this table depicts:

- The test number (N)
- The phase at which the test was added to the pile (TP),
- The situation under test (Situation),
- The test description(Description),
- The results of the final test (RFT).

In the table the term Supported Medical Device is encompasses all medical devices listed in the Requirement analysis document.

Table B.1: Test Pile

N	TP	Situation	Description	RFT
1	1	Connection of a Draeger Ventilator to the MD network	Does the system correctly instantiates the communication protocol?	yes
2	1	Connection of a Draeger Ventilator to the MD network	Does the system maintain a stable flow of data from the MD?	yes
3	2	Connection of a Servo-i Ventilator to the MD network	Does the system correctly instantiates the communication protocol?	yes
4	2	Connection of a Servo-i Ventilator to the MD network	Does the system maintain a stable flow of data from the MD?	yes
5	2	Connection of a Fresenius Infusion Pump to the MD network	Does the system correctly instantiates the communication protocol?	yes
6	2	Connection of a Fresenius Infusion Pump to the MD network	Does the system maintain a stable flow of data from the MD?	yes

7	2	Connection of a Vista BIS Monitor to the MD network	Does the system correctly instantiates the communication protocol?	yes
8	2	Connection of a Vista BIS Monitor to the MD network	Does the system mantain a stable flow of data from the MD?	yes
9	3	A Supported MD is connected to a port where a probing session instantiated	Does the system correctly detect the MD	yes
10	3	A Supported MD is connected to a port where a probing session instantiated	Does the system initialize a Data Session	yes
11	3	When a Data session is isntantiated	Does the system start generation Data objects?	yes
12	3	When a Data session is isntantiated	Is the probing session in that port properly terminated?	yes
13	3	When a Data object is created	Is the solution capable of inferring the bed that Data object belongs to?	yes
14	3	When a Data object is created	Does the Data buffer recieve the Data Object?	yes
15	3	When communications with a device are ceased.	Does the solution try to reestablish communication?	yes
16	3	When the communication with a device cannot be re-established.	Does the solution delete the device session?	yes
17	3	When the communication with a device cannot be re-established.	Does the solution Start a new Probing session on that port?	yes
18	3	When the communication with a device is re-established.	Does the Session resume the acquisition of data?	yes
19	3	When a Data Object reaches its life-span limit	Is that Data Object deleted?	yes
20	4	When data objects are retrieved by one subscriber	Are the requested Data objects to send back to the subscriber	yes

21	4	When data objects are retrieved by one subscriber	Does the subscriber correctly generate HL7 messages with MD data	yes
22	4	When na HL7 message is generated	Is it correctly stored in a message buffer	yes
23	5	When a new connection is recieved form na external application	Does the system correctly instantiate a new subscriber?	yes
24	5	When a known subscriber sends a valid query message to the HL7 broker	Does the solution update the configuration for that subscriber?	yes
25	5	When a known subscriber sends a valid query message to the HL7 broker	Does the solution send na ORF message responding?	yes
26	5	When a known subscriber sends an invalid query message to the HL7 broker	Does the solution issues a NAK message?	yes
27	5	When a known subscriber fails to send an ACK message to the HL7 broker	Does the solution resend that message?	yes
28	5	When a valid HL7 message is correctly generated.	Does the requesting application receive the message?	yes
29	5	Whens an external application sends an end of communication message.	Is the data subscription of the subscriber deleted?	yes
30	5	When relevant data to a given subscriber is made available in the data buffer	Does the solution generate a valid HL7 Message?	yes
31	5	When the Broker loses connection to the subscriber	Does the solution keep storing the data subscribed in the Buffer?	yes
32	5	When the Broker loses connection to the subscriber	Does the solution mantain that subscriber active?	yes
33	5	When the connection to a subscriber is re-established.	Does the solution send all the corresponding data stored in the buffer to that subscriber?	yes

34	5	When the connection to a subscriber cannot be re-established.	Does the solution terminate the session with the subscriber and releases the ip?	yes
35	F	When operating in a real situation	Is the solution capable of handling every supported MD in thata unit?	yes
36	F	When operating in a real situation	Is the solution capable mantainning working subscribers?	yes
37	F	When operating in a real situation	Is the solution stable for at least 48 hours?	yes
38	F	When operating in a real situation	Does the solution use less than 15% of CPU?	yes
39	F	When operating in a real situation	Does the solution use less than 200 Mb of RAM memory?	yes
40	F	When the system crashes or improperly shuts down	Is the solution capable of restoring its function?	no
41	F	When an external application sends more than 10 corrupted messages in one minute.	Does the solution block that connection?	no

Appendix C

HL7 Integration Document

1. Introduction

The purpose of this document is to describe the computer interface provided by B-Sharer with other devices and applications. It is intended for programmers with knowledge of the standard medical communication protocol HL7 version 2.4.

B-Sharer acts as the central data acquisition unit that stores and redistributes all medical device data generated in a medical unit. This middleware application interfaces with external devices via a broker that acts as a TCP/IP server.

This interface will only share information relative to data from medical devices. No personal information from patients is handled.

2. Communication Protocol

The external device must act as master and take the initiative when starting the communication session.

B-Sharer's broker works with a subscriber system, where new clients generate new subscriptions. A new subscription is created any time a device first establishes connection with the broker. This subscription will remain active until the broker gets no response from a client for 30 minutes. If a client loses connection with the broker the subscription will remain active and the client can reconnect and retrieve all the information still on the subscription buffer.

After the subscription is made, the client will have to issue a Query message (QRY^R02) that will configure the data response messages. Subsequent Query messages may be issued to reconfigure, update the content on data response messages or unsubscribe the service. The format of the query messages is further explained in the Chapter 3 of this document.

When the broker receives a valid query message, it will begin to periodically issue ORU^R01 messages containing the medical data. The format of the ORU^R01 messages is further explained in the Chapter 4 of this document.

3. The Query Command

The broker does not transport nor handles any patient information. As such it will only be possible to subscribe for medical data relative to a bed. If there are no medical devices generating data, the query will still be valid, but no data response messages will be generated.

The broker will expect messages of the type QRY^R02 with a format compatible with the 2.4 version of the HL7 Standard. This message is composed by three segments:

- Header (MSH segment)
- Original-style query definition (QRD)

- Original-style query filter (QRF)

The expected format of the header segment is the following:

MSH|^~\&|<Receiving application>||<Sending application>||<date>||QRY^R02|<Message ID>|P|2.4|

Field	Hl7 data format	Description
Receiving application	String	Name of the application the application that will receive the message.
Receiving application	String	Name of the application the application that is sending the message.
Message date	HL7 date	The timestamp of the message creation.
Message ID	Integer	The identification number of the message (Can't be repeated).

The format of the QRD segment is the following:

QRD|<query date>|R|I|<Query ID>|||1|<bedID>|OTH|I<MedicalUnit Code>^<MedicalUnit Name>||T

Field	Hl7 data format	Description
Query Date	Hl7 Date	Timestamp for the query creation (can be the same as the <Message date> of the header).
Query ID	Integer	Identification number of the query.
Bed ID	Integer	Identification number representing the bed of the medical unit witch the query is referred to. (According to the standard this field is referred to the patient ID, but here we use the bed ID)
Medical Unit code	Integer	The numerical code that represents the medical unit
Medical unit name	String	The name of the medical unit where the bed is located

The format of the QRD segment is the following:

QRF|<Bed ID>:Bed|||||||<Data Retrieval Mode>^Q<Interval>S^^^^^< Data type code >

Field	Hl7 data format	Description
Bed ID	Integer	Same as the Bed ID from the QRD segment (*)
Data retrieval mode	Integer	This field defines the data generation mode. There are two possible modes, the instantaneous mode and the continuous mode. (**)
Interval	Integer	This value defines the interval (in seconds) in between Hl7 messages generated by the broker
Data type code	String	This field can take one of two values: "ND" for Numeric Data or "RT" for Real Time data

(*) The bed number must be a two byte integer. The addition of a "-" character (0x2D) before the bed number will unsubscribe all message generation for that bed. The broker only accepts information of one bed per message, therefore if data for more than one bed is needed, that many query messages must be issued.

(**)To enable the instantaneous mode the code “1” must be inserted. In this mode the broker will only generate messages with the most recent data for all the equipments subscribed. This mode can only be used to send numeric data. The continuous mode will be enabled with the code “2”. This mode can be used for numeric data and real-time data. It will send all the data generated since the last message issued.

To completely unsubscribe the service the “-” character must be inserted in the Bed ID field, but now without any bed number. In this situation all remaining configuration information will be ignored.

A complete example of a query message can be seen in the figure below:

```
MSH|^~\&|B-ICUCare||B-Sharer||20140429173745||QRY^R02|0000001|P|2.4
QRD|20140514143454|R|I|BQ01|||1|01|OTH|24050^ICU NeuroCritics Unit HSJ||T
QRF|01:Bed|||||||1^Q10S^^^^^ND
```

In this example the client intends to subscribe data from bed 01 of an ICU unit with code 24050. This data will be numeric and sent in the instantaneous mode every 10 seconds.

These query messages will have its ORF^R04 response acknowledging the query.

```
MSH|^~\&|<Receiving application>||<Sending application>||<date>||ORF^R04|<Message ID>|P|2.4|
MSA|AA|<QRY^R02 ID>
QRD|<query date>|R|I|<Query ID>|||1|<bedID>|OTH|I<MedicalUnit Code>^<MedicalUnit Name>||T
OBR|1|||^Subscription
OBX|1|NA|^Beds||<Subscribed Beds>|I|||||F
```

The QRY^R02 ID is the id of the query message that originated this ORF^R04 message. The QRD segment is the same as the one in the query message. The Subscribed Beds field contains an array with all the beds currently subscribed.

A complete example of an ORF^R04 message can be seen in the figure below:

```
MSH|^~\&|B-Sharer||B-ICUCare||20140429173747||ORF^R04|000002|P|2.4
MSA|AA|0000001
QRD|20140514143454|R|I|BQ01|||1|01|OTH|24050^ICU NeuroCritics Unit HSJ||T
OBR|1|||^Subscription
OBX|1|NA|^Beds||[1 2 3]||||||F
```

This message is the response the Query presented as an example. According to this response the external application had already subscribed bed 2 and 3 in previous queries.

4. Data Response Messages

All Data response messages issued by the broker will follow the structure of the ORU^R01 messages of the as described in the 2.4 version of this standard. These messages are composed of three different sections:

- The header which is composed of 1 segment
- The Patient identification, composed by 3 segments
- The Order observations composed of a variable amount of segments

The expected format of the header section is the following:

MSH|^~\&|<Receiving application>||<Sending application>||<date>||ORU^R01|<Message ID>|P|2.4|

Field	Hl7 data format	Description
Receiving application	String	Name of the application the application that will receive the message.
Receiving application	String	Name of the application the application that is sending the message.
Message date	HL7 date	The timestamp of the message creation.
Message ID	Integer	The identification number of the message (Can't be repeated).

The 3 segments that compose the Patient Identification section are:

- Patient Identification Segment (PID)
- Patient Visit 1 (PV1)
- Patient Visit 2 (PV2)

PID|||<BedID>||Smith^John
PV1|<BedID>|I
PV2|<Point of Care>^<Room>^<BedID>^<Facility>

Field	Hl7 data format	Description
BedID	Integer	Identification number representing the bed of the medical unit witch the query is referred to. (According to the standard this field is referred to the patient ID, but here we use the bed ID)
Point of Care	String	The name of the Service The patient is currently in (ex. Neurocritical)
Room	String	The name of the Room The patient is currently in (ex. ICU or Internment)
Facility	String	The name of the Hospital the patient is Currently in.

The last section of the Data response message is where the actual data from the medical devices is located. This section is composed by a variable number of the following segments:

- Observation Request (OBR)
- Observation Result (OBX)
- Notes and Comments (NTE) (Optional)

OBR|<OBR Number>|||<Equipment Code>^<Equipment Name>|||<Data Timestamp>

Field	Hl7 data format	Description
OBR Number	Integer	Sequence number of this OBR segment in this message
Equipment Code	Integer	Numerical code representing the medical Device
Equipment Name	String	The name of the medical device

Data Timestamp	HL7 Date	The Timestamp of the generation of the data related to this device

OBX|<OBX #>|<Data Type>|<DV Code>^<DV Name>||<Data>|<Unit>^<U Name>^SI|||||F|<Date Last>||<Date>

Field	Hl7 data format	Description
OBX #	Integer	Sequence number of this OBX segment relative to its corresponding OBR segment
Data Type	String	Code representing the Type of data present on the Data field. It can take the values of “NA” for Numeric Array, “NM” for numeric and “ST” for String
DV Code	Integer	The Numerical Code representing the Data variable featured in this segment (See Section Data Codes for a list of all possible codes)
DV Name	String	The name representing the Data variable featured in this segment (See Section Data Codes for a list of all possible names)
Data	Variable	The data value formatted as stated in the Data Type field
Unit	String	Abbreviation of the name of the unit the in the International System
U Name	String	Full name of the unit in the International System
Date Last	HL7 Date	The Timestamp of the generation of the last data with the same code for the same device
Date	HL7 Date	The Timestamp of the generation of the data present in this segment

NTE|< NTE Number >||<Additional Information>

Field	Hl7 data format	Description
NTE Number	Integer	Sequence number of this NTE segment relative to The preceding OBX segment
Additional Information	String	String used to pass additional information relative to the preceding OBX segment. (In this version of the broker it is only used to pass the Sampling Frequency used in the acquisition of numeric Arrays.)

For each subscribed medical device an OBR Segment is generated in the message. Following this segment all its corresponding OBX segments are inserted. Any necessary NTE segments are inserted after its corresponding OBX segment.

Two complete examples of ORU^R01 messages are presented. The first is relative to a numeric data message and the second is relative to a Curve data message.

```
MSH|^~\&|B-Sharer||B-ICUCare||20140101010120||ORU^R01|MSG-000001|P|2.4
PID|||01||Smith^John
PV1|01|I
PV2|Neurocríticos^ICU^1^HSJ
OBR|1|||01^Servo-I|||20140101010105
OBX|1|NM|503^Mechanical Breath Rate||20|^1/min||||F
OBX|2|NM|280^Expiratory Tidal Volume||5|^L|||||F
```

```
MSH|^~\&|B-Sharer|ICU NeuroCritics Unit HSJ|B-Sharer|ICU NeuroCritics Unit
HSJ|20140825104803||ORU^R01|1408963682877|P|2.4||1
PID|||1||Smith^John
PV1|||I|^^1
PV2|Neurocríticos^ICU^1^HSJ
OBR|1|||^Vista|||20140825104800
OBX|1|NA|4051^EEGC||[0.2,-0.15,0.15,0.3,0.15,0,0,0,0,0,0,0,0,0.05,0.2,0.15,-0.2]|mv|||||F
NTE|01||128 Hz
OBX|2|NA|4052^EEGC||[6.4,-0.15,0.15,0.3,0.15,0,0,0,0,0,0,0,0,0.05,0.2,0.15,-0.2]|mv|||||F
NTE|01||128 Hz
```

All ORU^R01 must be answered with an HL7 ACK message. If this message is not received by the broker all the data will be inserted in the next ORU^R01 message. The structure of this message is as follows:

MSH|^~\&|<Receiving application>||<Sending application>||<date>||ACK|<Message ID>|P|2.4|
MSA|AA|<ID ORU^R01>

Field	Hl7 data format	Description
Receiving application	String	Name of the application the application that will receive the message.
Receiving application	String	Name of the application the application that is sending the message.
Message date	HL7 date	The timestamp of the message creation.
Message ID	Integer	The identification number of the message (Can't be repeated).
ID ORU^R01	Integer	The Message ID of the ORU^R01 message to be acknowledged

5. Supported Medical Devices

B-Sharer supports the following medical devices:

Generic Device Name	Equipment Code	Equipment Name
Ventilators	01	Maquet Servo-i
	02	Maquet Servo-s
	03	Dräger Evita 4
Infusion pumps	04	Orchestra Fresenius
BIS Monitors	05	Aspect Medical Systems: Vista

6. Data Codes

Equipment Type	Data Code	Parameter Name	Data Type
Ventilator	503	Mechanical Breath Rate	Numeric

DataCodes	280	Expiratory Tidal Volume	Numeric
	279	Inspired Tidal Volume	Numeric
	119	Inspired Minute Volume	Numeric
	118	Expired Minute Volume	Numeric
	454	Peak pressure	Numeric
	110	Mean Airway Pressure.	Numeric
	103	Plateau Pressure	Numeric
	149	Inspired Oxygen Concentration	Numeric
	94	End-Tidal CO2 Concentration	Numeric
	431	Expiratory Resistance	Numeric
	433	Inspiratory Resistance	Numeric
	93	Static Lung Compliance	Numeric
	99	Peak Expiratory Flow	Numeric
	100	Peak Inspiratory Flow	Numeric
	433	Inspiratory Resistance	Numeric
	111	1: Inpired:Expired Ratio	Numeric
	92	Dynamic Lung CompliVitalSigns	Numeric
	359	Dynamic Lung Resistance	Numeric
	2010	PEEP total	Numeric
	107	Intrinsic PEEP Breathing Pressure	Numeric
	594	Spontaneous Respiration Rate	Numeric
Infusion Pump DataCodes	4000	Infusion Pump - Module	Numeric
	4001	Drug Name	String
	4002	Infused Volume	Numeric
	4003	Programmed Volume Bolus	Numeric
	4004	Remaining Volume	Numeric
	4005	Infused Volume	Numeric
	4006	Concentration	Numeric
	707	Infusion Pump Delivery Rate	Numeric
BIS Monitor DataCodes	5000	EEG Suppression Ratio L	Numeric
	5001	EEG Suppression Ratio R	Numeric
	5002	BIS Signal Quality Index L	Numeric
	5003	BIS Signal Quality Index R	Numeric
	5004	Bispectral Index L	Numeric
	5005	Bispectral Index R	Numeric
	4050	Raw EEG Curve Data	Numeric Array

Appendix D

MD Communication Potocol

In this appendix are described the proprietary communication protocols used by the MD with whom the solution will integrate with. These communication protocols are:

- The **MEDIBUS** protocol is used by the Draeger Ventilators,
- The **Agila Serial Export** protocol is used by the Fresenius infusion pumps,
- The **CEI** protocol is used by the Servo-I Ventilators
- The **VISTA Binary** protocol is used by the BIS monitor

D.1 MEDIBUS protocol

This communication protocol is used by all Draeger® ventilators to exchange data via RS-232 interfaces. Through this interface two types of message are exchanged, commands and response messages. A command is emitted by a device that expects to access data from another device. The response is emitted by the device that received a command message. Some response messages can possess embedded commands. Medibus consists of two independent protocols, the 'slow' and the 'fast' that can run simultaneously in the same communication line. The 'slow' mode is used to transport numerical data from the ventilator with a periodicity in the temporal scale of seconds using ASCII encoded messages with charecters ranging from the 0x00 until the 0x7f. The 'fast' protocol is used to transport real-time data from the ventilators. These protocols can run simultaneously, because the 'fast' protocol also uses ASCII encoded messages but ignores the charecters ¡ 0x80 used by the 'slow' protocol. In this work only the 'slow' protocol will be implemented.

D.1.1 Communication life-cycle

Whenever a ventilator is connected, it starts emitting the command 'initialize communication command' (ICC) periodically until a valid response to this command is received. When a response to the ICC command is receive, the ventilator issues a command to request the identification of the responding device. If the response to this command is valid the MEDIBUS protocol is active and the device is ready to communicate data in the 'slow' protocol. The protocol also defines commands to suspend, resume and abort message transmissions and commands to terminate the communication session. In this protocol two time-out mechanisms are implemented that terminate the communication if no messages are exchanged or if a complete response message is not received in a given time frame. In order to avoid time-out events when no messages need to be exchanged for long periods of time, the protocol allows special commands that don't require a response message to be issued in a pre-established frequency. If a time-out event occurs the communication is considered broken by the device and it restarts emitting ICC commands.

D.1.2 Message Structure

The structure of a command is presented in figure D.1. All command messages are initiated by a 1 byte header followed by the code of the command also with 1 byte. If the command requires an argument the information regarding the argument follows the command code and can have variable size. In order to check the validity of the message a checksum field with 2 byte is inserted in the message before the 1 byte terminator.

Header		Command Code		Argument		Checksum		Terminator	
1	1	1	2	N	N+2	2	N+4	1	N+5

FIGURE D.1: Generic MEDIBUS Command

A response message is also initiated with a 1 byte header followed by the echo of the command that originated the response. The actual response is encoded next in a field with variable size. Similarly to the command message, the response is also terminated by a checksum field and a terminator character. Some responses can also have no response field, for example the response to a control, unknown or corrupted command. The response to a corrupted command contains a special code instead of the command echo that informs the recipient of the response that the command received was corrupted. In figure D.2 a generic response message is presented.

Header		Command Code		Response		Checksum		Terminator	
1	1	1	2	N	N+2	2	N+4	1	N+5

FIGURE D.2: Generic MEDIBUS Response

The code used in the header of a message is different according to the message types but the terminator is always the same. The size of a message in this protocol can be N+5 byte if the message has an argument or a response field or 5 byte if not.

D.2 Agila Serial Export Protocol

This protocol is used to communicate with the infusion pump systems Orchestra Base Intensive and Orchestra Base Primea manufactured by Fresenius. Each infusion pump system is composed of a central module called a 'base' where 1-8 syringe or volumetric pumps can be assembled. All the internal communications between the modules and the base station is performed internally. External communication is performed via a full-duplex asynchronous RS-232 interface. Through this connection line the user can monitor both the base station and the pump modules.

D.2.1 Communication life-cycle

In order to establish a connection with an infusion pump system through this communication protocol, it is necessary to send a 'start of communication' command. The MD then replies with an acknowledgement massage and becomes ready to receive messages. If a message is stent before a communication is opened, the MD replies with an error message. The communication can be manually terminated using a 'end of communication' command.

All message exchange processes follow an acknowledgment protocol therefore, every time a message is received by any device, it must send an acknowledgment message informing the other device that he is ready to receive another message. As the communication is full duplex, it is possible to receive a command while sending a message, when this happens the device will only send the acknowledgment message after sending the message, therefore the character code for the acknowledgment command must not be used in other messages. If a device receives a corrupted message, an error message is sent instead of the acknowledgment message.

It is possible to implement time-out mechanisms in the communication. When this feature is active, the systems periodically exchange special characters. This exchange

process is independent of the message exchange process and can therefor occur simultaneously. If this character is received in the middle of a message, the receiving device must ignore it for the message processing context. If a time-out is verified all the communications are ceased.

D.2.2 Message Structure

All the messages exchanged while using this communication protocol have a similar structure. The messages have a 1 byte header followed by a 1 byte character that identifies the module number, or base station, that the message is related to. The message is then followed by a variable length field that contains the message body. This message body can contain command codes with its respective arguments or response codes with associated with data. Similarly to the MEDIBUS protocol, the messages are ended with a 2 byte checksum field and a 1 byte termination character. In figure D.3 is presented the structure of a generic message. The size of a message can vary between 5 byte or N+5 byte depending on the size of the body of the message.

Header		Number		Message		Checksum		Terminator	
1	1	1	2	N	N+2	2	N+4	1	N+5

FIGURE D.3: Generic Agila Serial Export Message

D.3 CEI protocol

This protocol is used by the manufacturer MAQUET to interface the ventilators Servo-i and Servo-s with external equipments via an RS-232 serial interface. Here the external equipment must work as master and transmit commands to the MD in order to retrieve information.

In this protocol all of the respiratory parameters are stored in a different channel and the device supports 999 different channels. This protocol supports two modes of operation, the 'Basic' and the 'Extended' modes. The 'Basic' mode uses the first 100 channels and can be used to retrieve numeric values and buffered curve data from a limited number of parameters. The 'Extended' mode supports a wider range of numeric and curve parameters and it allows the external device to acquire real-time data. For this it uses the channels 100-999. Due to memory restrictions the number of curve data channels that can be sampled at the same time is restricted to 4.

D.3.1 Communication life-cycle

In order to initiate communication with the ventilator, the external equipment must issue a 'hello' command. The ventilator responds with a standard message and enters in the 'Basic' mode. This mode only accepts 'Basic' commands and a special command used to enter in the 'Extended' mode. The 'hello' command can also be used to return to the 'Basic' mode from the 'Extended' mode.

After establishing the communication in the desired mode, the external device must then subscribe the channels containing the data it intends to access. This subscription is valid for the rest of the session or until a new subscription is made. In order to access the subscribed data the external device must issue commands to read the data buffers of the ventilator.

D.3.2 Message Structure

The commands issued by the external device are different depending on the mode that is currently activated. In the 'Basic' mode the messages are initialized with the command code codded in 2 ASCII characters. Depending on the issued command the message can be followed by a variable amount of fields, each enclosing the value of an argument. These fields can also have variable lengths. All the messages are then terminated with a 1 byte termination character. The messages in the 'Extended' mode are similar to the ones above, but here the command codes are encoded in 4 ASCII characters instead of 2 and before the termination character, a checksum value of 2 byte must be inserted.

The structure of the response generated by the device can vary depending on the command received, but all the responses are composed by a number of variable sized fields, followed by the checksum value, if the 'Extended' mode is selected, and terminated by the same termination character.

In both communication modes an empty command can be issued, this message is composed solely by the termination character and is used primarily to check the connection.

The value of all parameters is transmitted in the form of numbers between 0000 and 9999. In order to access the real value, measured in engineering units, the transmitted value must be converted trough a formula that takes into consideration the scale factor associated with each parameter.

D.4 VISTA Binary protocol

This protocol is used by the BIS monitor manufactured by Aspect Medical Systems to communicate with external devices via a RS-232 serial interface. The BIS monitor can be interfaced using 2 different protocols, one denominated ASCII protocol and the other Binary protocol. The protocol that will be used in the communication with external devices must be firstly selected in the monitor's settings menu.

The ASCII protocol is the simplest one and is used to retrieve some processed EEG parameters. These parameters are assembled in records that can be retrieved by external devices using single character commands. Each record is composed of a variable number of 8 character fields separated by a 'vertical bar' character. All records are terminated using a 2 byte termination code.

The Binary protocol is more versatile as it can access a wider array of processed EEG data as well as raw EEG data. This protocol can be used in one of 2 modes, the 'Legacy Binary' that is mainly used to maintain compatibility with older systems and the 'VISTA Binary' that is used by the most recent devices and supports bilateral sensors. This operation mode must also be selected in the device's settings menu. The following section are related uniquely to the 'VISTA Binary' mode, as this will be the mode used I this work.

D.4.1 Communication life-cycle

As soon as the communication mode is selected in the device's settings menu, the device is ready to receive commands from external devices. These commands and the respective responses are exchanged via packets. Here 3 types of packets are defined, the data packets, the acknowledgement packet (ACK) and the negative acknowledgement packet (NAK).

Every time a data packet is received, the receiving device must issue a ACK packet in a given time frame to the sender, otherwise the sender will assume a time-out event. If the packet is corrupted a NAK packet must be issued. When a time-out occurs or a NAK is received, the external device must resend the command packet that originated the event. It is also unnecessary to send ACK or NAK packets to the monitor as it will always assume an ACK packet and will never repeat the information previously sent.

D.4.2 Message Structure

All the packets exchanged have a 3 layered structure. The first layer is the physical layer and is concerned with the transmission of data over the serial link. The second layer provides routing information that can be used for routing the message in a system with multiple processes. The last layer is the application layer and is concerned with the transmission of data.

The first layer is composed of 6 fields organized as in figure D.4. The first field of a packet is a 16 bit header. This header is followed by a 16 bit unique sequence determined by the monitor. This sequence is set to 0 every time the monitor is connected. The number of bytes of both the second and the third layers is inserted next in a field with 16 bit length. The layer 1 directive is inserted next and here is where the type of packet is defined (data, ACK or NAK). If the packet is of the type NAK or ACK the optional data field is ignored. If it is a data packet, the fields regarding the second and third layers are inserted in this field. In the end of each packet, a checksum with 16 bit length is inserted.

Header	Packet Sequence ID	# bytes of Opt. Data	Layer 1 directive	Optional Data (data from layer 2 and 3)	Checksum
16	16	16	16	variable	16

FIGURE D.4: First Layer of the VISTA Binary protocol packet

The second layer is composed of a field 32 bit long comprising a routing identifier that is used to identify a specific task or mailbox to which a command or the results of a command are to be directed. The last field of this layer is the data from the application layer with a variable size. In figure D.5 is presented the structure of the second layer.

Routing identifier	Application Data
16	variable

FIGURE D.5: Second Layer of the VISTA Binary protocol packet

The third and final layer is composed of a message ID field that contains the code of the command or the code of the response, this field is 32 bit long. The next field is the sequence number, the purpose of this field is similar to the Packet Sequence ID, but here a separate sequence number is kept for each different Message ID. The number of bytes of the message depended data is inserted next and has a size of 16 bits. The last

parameter, the Message Dependent Data, is where the parameters or the EEG data are inserted, depending if the packet is a command or a response. In figure D.6 is presented the structure of the third layer.

Message ID	Sequence Number	Length	Message Dependent Data
32	16	16	variable

FIGURE D.6: Third Layer of the VISTA Binary protocol packet

Bibliography

[1] West Health Institute. The Value of Medical Device Interoperability accessed in 2014-01-28. URL https://s3.amazonaws.com/wwhi.org/interop/WHI-The_Value_of_Medical_Device_Interoperability.pdf.

[2] B-Simple. accessed in 2014-01-28. URL http://www.b-simple.pt/.

[3] University of Cambridge Enterprise. ICM+® Software for Brain Monitoring in Neurological Intensive Care accessed in 2014-01-28. URL http://www.enterprise.cam.ac.uk/industry/licensing-opportunities/icm-software-for-brain-monitoring/.

[4] K. Lesh, S. Weininger, and J. M. Goldman. Medical device interoperability – assessing the environment. *Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, pages 1–10, 2007. URL ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04438159.

[5] Health Level 7. About HL7 accessed in 2014-01-28. URL http://www.hl7.org/about/index.cfm?ref=nav.

[6] World Health Organization. International Classification of Diseases (ICD) accessed in 2014-01-28, . URL http://www.who.int/classifications/icd/en/.

[7] B. Courtney. An investigation into the use of hl7 clinical document architecture as a standard for discharge summaries in ireland. *Master Thesis*, 2011. URL https://www.scss.tcd.ie/postgraduate/health-informatics/assets/pdfs/An%20investigation%20into%20the%20use%20of%20HL7%20Clinical_BC.pdf.

[8] Corepoint Health. The HL7 Evolution accessed in 2014-01-28. URL https://www.corepointhealth.com/sites/default/files/whitepapers/hl7-history-v2-v3.pdf.

[9] International Health Terminology Standards Development Organization. SNOMED CT accessed in 2014-01-28, . URL http://www.ihtsdo.org/snomed-ct/.

[10] Institute of Electrical and Electronics Engineers. SNOMED CT accessed in 2014-01-28. URL http://standards.ieee.org/.

[11] I. Martínez, J. Fernández, and M.Galarraga. Implementation experience of a patient monitoring solution based on end-to-end standards. *Proceedings of the 29th Annual International Conference of the IEEE EMBS Cité Internationale, Lyon, France*, pages 6425–6429, August 2007. URL http://www.ncbi.nlm.nih.gov/pubmed/18003493.

[12] M.Galarraga, I. Martínez, and P. Toledo. Review of the iso/ieee 11073 - pocmdc standard for medical device interoperability and its applicability in home and ambulatory telemonitoring scenarios.

[13] Continua Health Alliance. About the Alliance accessed in 2014-01-28. URL http://www.continuaalliance.org/.

[14] iMD soft®. MetaVision for Critical Care accessed in 2014-08-24. URL http://www.imd-soft.com/critical-care.

[15] Optum®. Critical Care Solutions accessed in 2014-08-24. URL http://www.optum.com/providers/clinical-performance/critical-care.html.

[16] ALERT®. ALERT®Patient Data Management System accessed in 2014-08-24. URL http://www.alert-online.com/pdms.

[17] Capsule®. Capsule Solutions accessed in 2014-08-24. URL http://www.capsuletech.com/solutions.

[18] True Process®. Medical device connectivity solution accessed in 2014-08-24. URL http://trueprocess.com/products/medical-device-connectivity-solution.