DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Development of an Embedded System for a Platelet Illumination Device

Relatório submetido à Universidade Federal de Santa Catarina

como requisito para a aprovação na disciplina

DAS 5511: Projeto de Fim de Curso

Tatiana Beber Kassick

Florianópolis, February 2016

Development of an Embedded System for a Platelet Illumination Device

Tatiana Beber Kassick

Esta monografia foi julgada no contexto da disciplina
DAS5511: Projeto de Fim de Curso
e aprovada na sua forma final pelo
Curso de Engenharia de Controle e Automação

Prof. *Leandro Buss Becker*

_____
Assinatura do Orientador

Banca Examinadora:

Vincent Krawczyk
Orientador na Empresa

Prof. Leandro Buss Becker
Orientador no Curso

Prof. Rodrigo Castelan Carlson
Avaliador

Lucas Caldeira de Oliveira
Rafael Bergamin Gonçalves Borges
Debatedores

Acknowledgements

Firstly I would like to express my gratitude to Mr. Thierry VERPOORT, the Automation's service director, for giving me the opportunity of joining his department as an intern for this last six months.

I am also grateful to Mr. Pierre-Eloi BONTINCK, the Research and Development manager, for welcoming me into his team and entrusting me with the Macotronic UV project.

To my supervisor Mr. Vincent KRAWCZYK my most sincere thanks for sharing his expertise and guiding me throughout the project, for his patience, advices and overall mentoring, which immensely enrichened the whole internship experience.

I convey my deep appreciation for my advisor Prof. Leandro B. BECKER and all his counseling and insightful observations, specially during the writing of this document.

My remaining fellow co-workers, the amazing team at Macopharma's Automation R&D department – Alan CADIOT, Anthony MARQUES, Arnaud CHAVATE, Aurélien REVELEAU, Kévin MOISAN, Michel COLIN, Nicolas COASNE and Sébastien VANHELST – also have my thanks for their warm reception and valuable help whenever I needed some.

Finally, I would like to thank my parents, for their neverending support and encouragement, and my partner for his continous and unwavering companionship during this venture.

Resumo

Este documento relata os trabalhos realizados dentro do escopo do projeto de conclusão do curso de Engenharia de Controle e Automção da Universidade Federal de Santa Catarina. Este projeto foi desenvolvido pela aluna Tatiana Beber Kassick durante seu estágio efetuado no seio da empresa Macopharma que projeta e produz equipamentos e dispositivos médicos.

No setor de Pesquisa e Desenvolvimento em Automação da divisão de Transfusão, o projeto consistia em implementar um novo sistema embarcado para a máquina de iluminação de plaquetas sanguínias: a Macotronic UV. Esse aparelho permite a execução, o controle e o monitoramento do processo THERAFLEX UV-Platelets.

Plaquetas são componentes do sangue humano que, graças a seus efeitos coagulatórios, servem principalmente para estancar sangramentos e reparar vasos sanguínios danificados. Assim, transfusões de plaquetas são um tratamento altamente utilizado para prevenir hemorragias em pacientes com baixa concentração destes elementos.

A partir de uma doação de sangue pode-se obter um concentrado de plaquetas ao separá-las dos outros componentes sanguíneos – glóbulos brancos e vermelhos – através de centrifugação. No entanto esse concentrado pode conter agentes patogênicos, como bacterias e virus, advindos do sistema do doador o que representa risco de contaminação e infecção para o paciente receptor. Glóbulos brancos remanescentes também podem causar reações adversas prejudiciais.

Levando em consideração esses riscos e a fim de evitá-los ou ao menos reduzí-los signifivamente, diversas ténicas de inativação de patógenos já foram desenvolvidas. O processo THERAFLEX, elaborado pela empresa Macopharma, é um exemplo inovante nessa área. Ele utiliza luz ultravioleta de onda curta UVC (especificamente 254 nm de largura de onda), também chamada de luz germicida, para atingir diretamente os ácidos desoxirribonucleico e ribonucleico (ADN e ARN) patogênicos sem, todavia, afetar a estrutura plaquetária, importante para obtenção

de uma transfusão eficaz e bem sucedida. Para uma exposição uniforme, o processo THERAFLEX UV combina à iluminação um movimento rotatório da bolsa de concentrado de plaquetas, atingindo assim melhores resultados na inativação de patógenos.

O processo é automatizado pela máquina Macotronic UV, composta por uma placa-mãe, uma placa de entradas e saídas analógicas gerenciada por um microcontrolador, um pirômetro, quatro fotodiodos, seis lâmpadas ultravioletas, motores, relés e ventiladores. Seu sistema embarcado compreende uma interface homem-máquina, um software para o microcontrolador e um protocolo de comunicação entre esses dois componentes.

Apesar de já existir um protótipo deste dispositivo, uma revisão do seu sistema embarcado se fez necessária na busca por otimização do controle do processo, por um aumento de precisão durante a iluminação, por aprimoramentos em nivel de conforto para os usuários e para elevar a capacidade de manutenção dos códigos. O projeto descrito neste documento se dividiu, então, em três etapas: desenvolvimento de uma nova interface gráfica, alterações no programa do microcontrolador e adaptação do protocolo de comunicação às mudanças realizadas.

Abstract

This document describes the development of a new embedded system for Macopharma's Macotronic UV illumination device. Consisting of a human-machine interface application, a microcontroller software and a communication protocol between the two, this system aims to provide intuitive features and fully monitored process control, ensuring high level precision during the THERAFLEX UV-Platelet procedure to improve platelet quality and allow health providers to offer purer and safer platelet transfusion treatments.

Sumary

List of Illustrations

List of Tables

List of Abbreviations

AIDS – Acquired Immune Deficiency Syndrome

ASCII - American Standard Code for Information Interchange

CPU – Central Processing Unit

DLL – Dynamic-Link Library

DNA – Deoxyribonucleic Acid

ECP – Extracorporeal Photo chemotherapy

FDA – Food and Drugs Administration

GUI – Graphic User Interface

HIV – Human Immunodeficiency Virus

HMI – Human Machine Interface

IDE – Integrated Development Environment

INI – Initialization (file extension)

LCD – Liquid Crystal Display

MVC – Model-View-Controller

OS – Operating System

PIT – Pathogens Inactivation Technology

PSL – Platelet Storage Lesion

RNA – Ribonucleic Acid

UV – Ultraviolet: light spectrum with wavelength from 100 to 400 nm

UVA – wavelength range from 315 to 400 nm

UVB – wavelength range from 280 to 315 nm

UVC – wavelength range from 200 to 280 nm

XML – Extensible Markup Language (file extension)

Chapter 1. Introduction

Platelets, also called thrombocytes, are a type of blood cell produced in the bone marrow whose main function is to stop bleeding and repair damaged blood vessels. Platelet transfusion is widely used to treat or prevent bleeding in patients with low platelet counts (due to diseases such as leukemia) or who have suffered massive blood loss (in case of injury, major operation or severe infection).

When blood is donated, a platelet concentrate can be obtained by separating the platelets from the other blood cells, such as erythrocytes and leukocytes, respectively known as red and white blood cells. The platelet concentrate would then be transfused into another patient. However, as with every transfusion procedure, there are risks of infection if the originally donated blood was contaminated by bacteria and/or viruses. Furthermore there is the possibility of adverse reactions and immune responses if there are remaining leukocytes present in the platelet concentrate.

To minimize these risks and extend shelf life of the platelets concentrates, many pathogen inactivation technologies (PITs) have been developed. Most of the PITs are based on adding a photosensitive compound to the platelet concentrate and activating it with ultraviolet lights in the UVB and UVA spectral regions, respectively wavelengths of 280 to 315 nm and 315 to 400 nm. This photosensitive compound, when activated, targets the nucleic acid of bacteria, viruses and leukocytes irreversibly damaging their DNA/RNA and consequently inactivating these pathogens. Although being an effective procedure, it requires a secondary filtering process to eliminate the added photoactive chemicals so as not to cause any toxicity in the platelet concentrate.

The THERAFLEX UV-Platelets PIT, developed by Macopharma Enterprises, has the innovative advantage of using ultraviolet light alone to inactivate pathogens. Without the addition of any photosensitive agent, the filtering phase can be eliminated and the transfusion patients are spared the additional risk of exposure to a toxic element.

Based on the application of short-wavelength ultraviolet light (UVC: wavelength of approximately 254 nm), which is absorbed directly by the DNA/RNA nucleobases of the pathogens contained in the platelet concentrate, the THERAFLEX UV-Platelets is a rather simple procedure, proven highly effective in inactivating bacteria, viruses and residual white blood cells, while preserving platelet functions and quality. Optimal delivery of UVC light is attained when the illumination is combined with the orbital motion of the platelet concentrate bag which leads to uniform exposure. The procedure is performed by the Macotronic UV illumination machine that ensures a micro-processor controlled and fully monitored treatment of the platelet concentrate.

In summary, the Macotronic UV illumination device consists of an aluminum and stainless steel based structure, an LCD touch screen, a front panel door, an ejectable tray to place and agitate the platelet concentrate bag, six UVC lamps, a Central Processing Unit (CPU) board to run the user interface software and a Rabbit® microcontroller to manage motors, sensors and relays that are also part of the device.

Within the Transfusion Division of Macopharma Enterprises, the goal of the project described in the present document was the development of a new embedded system for this equipment. Although the Macotronic UV and its embedded system already existed in a prototype version, the company decided it was time for a software revision to obtain control optimization, increase the illumination precision, improve user comfort and enhance maintainability.

.

1.1: Macotronic UV Project

The embedded system for the Macotronic UV illumination device is composed of a Human-Machine Interface (HMI) application, a microcontroller program and a communication protocol between them.

Originally programmed in C#, the software for the HMI application was entirely re-programmed in C++ because of the language's better performance, portability

between different operational systems and the general programming tools it provides (multiple inheritance, pointers, class structure and macros). The Qt cross-platform library framework was used for its recognizable graphic designer tools as well as its simplicity, flexibility and robustness when developing user interfaces and deploying them in several targets.

The existing microcontroller software was coded in Dynamic C: a C-like programming language for Rabbit®-based products. During this project the software was modified to include new features and solve known problems. In effect, the modifications extended to alter the operational procedure in order to convey more autonomy to the microcontroller. Whereas the previous version of the system handled the HMI and microcontroller in a master-slave capacity, the new system envisioned a more collaborative relationship.

This significant increase in independence gives the microcontroller true dominium over its charges (motors, sensors and other actuators). It also transfers the responsibility to constantly verify the components states and identify hardware problems and/or components failure from the HMI to the microcontroller's software. The new system dynamics leaves the HMI application to control more user based actions and reduces the back-and-forth of unnecessary commands.

Nevertheless, there are still commands and requests that must be sent from the HMI application to the microcontroller and replies to be returned. To establish a line of communication compatible with the new operating mode the existing communication protocol was adapted.

1.1.1: HMI Application

A new graphic design was developed for the HMI in order to render the application more user-friendly. Its visual aspects also take into account a more standardized design so as to allow users to be familiar with many of Macopharma's blood safety machines.

Providing the users with cycle management and monitoring, the HMI is the direct link with Macopharma's clients and as such it is an important representative of the enterprise's identity. The application has to be intuitive and simple to use while awarding the user with sophisticated screens and a wide variety of essential inlaid features. Launching the illumination cycle is, of course, the most important action a user can perform with the application.

Once the cycle is launched the HMI displays on the screen information regarding the delivered light intensity, the platelet unit's temperature, the agitation speed of the internal tray and the cycle's duration. It also warns the user in case of errors before, during and after the illumination treatment. The application is also responsible for generating all the pertaining files, labels and reports at the end of each and every cycle. This ensures traceability, an attribute of great value in the health and blood products sector.

1.1.2: Microcontroller Software

As previously stated, the microcontroller software manages the machine's actuators and sensors. While the original software performed all actions separately when directly commanded by the HMI application, the new program has a more self-governing nature. In other words, in the former version, for each action to be executed there had to be a command issued from the HMI (e.g. open front panel door or turn ultraviolet lamps on), whereas in the modified version a single command can set off a succession of actions managed entirely by the microcontroller software.

One example of the implications of the past dynamics is that an illumination cycle represented a long list of commands and responses, with no sequencing of the clearly related actions. This affected the machine's performance and contributed to increase communication derived errors. Provided that an illumination cycle has a very well defined and sequenced set of steps to follow and that neither the steps nor the sequence varies from cycle to cycle, there was no reason not to optimize the operational procedure.

The optimization enables the execution of all the actions pertaining to an illumination cycle after a single “Start Cycle” command is sent by the HMI. Upon reception of the command the microcontroller will close the front panel door, perform pre-cycle hardware tests, activate the tray agitation motor, turn UVC lamps on and execute regular hardware tests until the end of the cycle. The microcontroller’s program can identify on its own the end of the cycle by verifying the delivered light intensity. Once it reaches the ideal dose the treatment is considered completed and the software initiates the chain of events that determines the end of an illumination cycle, notably turning the ultraviolet lamps off, deactivating the tray agitation motor, opening the front panel door and ejecting the tray.

Other modifications were made to better adapt the program to the new operational mode and to improve code readability and maintainability.

1.1.3: Communication Protocol

Considering the aforementioned requirements, the communication protocol had to accommodate several new commands. These commands and their overall structure were defined and documented. The protocol remained compatible with the previous versions of the microcontroller software so that once the prototype machines (that are in the field running clinical trials) are updated any discontinuity issues will be prevented.

1.2: Document Structure

The remainder parts of this document are organized as follows:

- Chapter 2: presents Macopharma enterprises, its core divisions and associated products. The Research and Development activities within the Automation department are briefly described.

- Chapter 3: explores health and biological concepts and the blood products used along the present work. The THERAFLEX UV-Platelets procedure is explained to contextualize the problematic of the project.
- Chapter 4: analyzes the programming tools, techniques and practices used for the development of the HMI application. The software architecture is examined as well as its individual modules. The implementation of the interface's graphical elements is detailed.
- Chapter 5: discusses the modifications made to improve the Macotronic UV device's operating mode. The implementation of the automatic internal hardware tests is briefly approached. The commands included in the existing communication protocol and its basic structure are introduced to the reader.
- Chapter 6: highlights the global results of the project and constructs a personalized opinion about the work done.

Chapter 2. **Macopharma Enterprises**

Derived from the Vandeputte-Marchand wool product group, Macopharma was created as a result of a sudden need for diversification brought upon when new competitors based in Asia and North-Africa started to dominate the French textile market in the 1970s. At the same time, the blood transfusion field in France was undergoing a revolutionary transformation as the previously used breakable glass bottles were being replaced by durable plastic blood bags conceived by the American doctors Carl Walter and William P. Murphy, in the early 50s.

Taking under consideration the textile sector's difficult situation and in view of the opportunities arisen in the blood transfusion department, the Vandeputte-Marchand group decided to invest in gathering medical and pharmaceutical professionals whose expertise would allow the production of blood bags.

Macopharma Enterprises was, thus, officially founded in 1977 in Tourcoing France. The company profited from an accelerated growth rate and was soon able to innovate by developing and perfecting filters to reduce the presence of leukocytes in the blood bags. In parallel, it also started an intravenously injectable solutions line and had 620 thousand infusion bags leaving the plant in the first year of production.

The next step was expansion. The 1990s were marked with product sales throughout Europe followed by the creation of several international subsidiaries. In 2003, the FDA's (Food and Drugs Administration) approval of one of Macopharma's leukocyte reduction devices opened the door into the American market. New production sites were inaugurated in France, Poland and Tunisia to accommodate the growing demands with larger industrial output.

When the World Health Organization issued an alert regarding the bird flu pandemic, in 2005, Macopharma, with its knowledge of high volume production, launched a new area of activity: respiratory masks. The production of masks intensified, going from 90,000 to 4 million units per week, following the announcement of the type A flu in 2009. In another front, as transfusion medicine evolved towards new cell treatments, Macopharma proceeded to diversify its

activities by developing biotherapy systems to contribute in the fields of regenerative medicine, transplantation and tissue therapy. With this initiative the four pillar divisions of the company were defined.

With a global turnover of 171.3 million euros in 2014, over 2400 employees around the world and product presence in 88 countries [1], Macopharma is now a prominent international company, leader in transfusion, biotherapy, hospital devices and respiratory masks. In the coming years, it aims to reinforce its business and to stay attentive to new industrial and marketing opportunities which could contribute to its ongoing expansion, especially in high-growth areas like Brazil. Effectively, the group plans to extend its activities in Brazil by sharing knowledge in the blood product field and establishing partnerships with Brazilian companies, such as the local manufacturer of blood transfusion kits JPFarma, based in Ribeirão Preto, São Paulo.

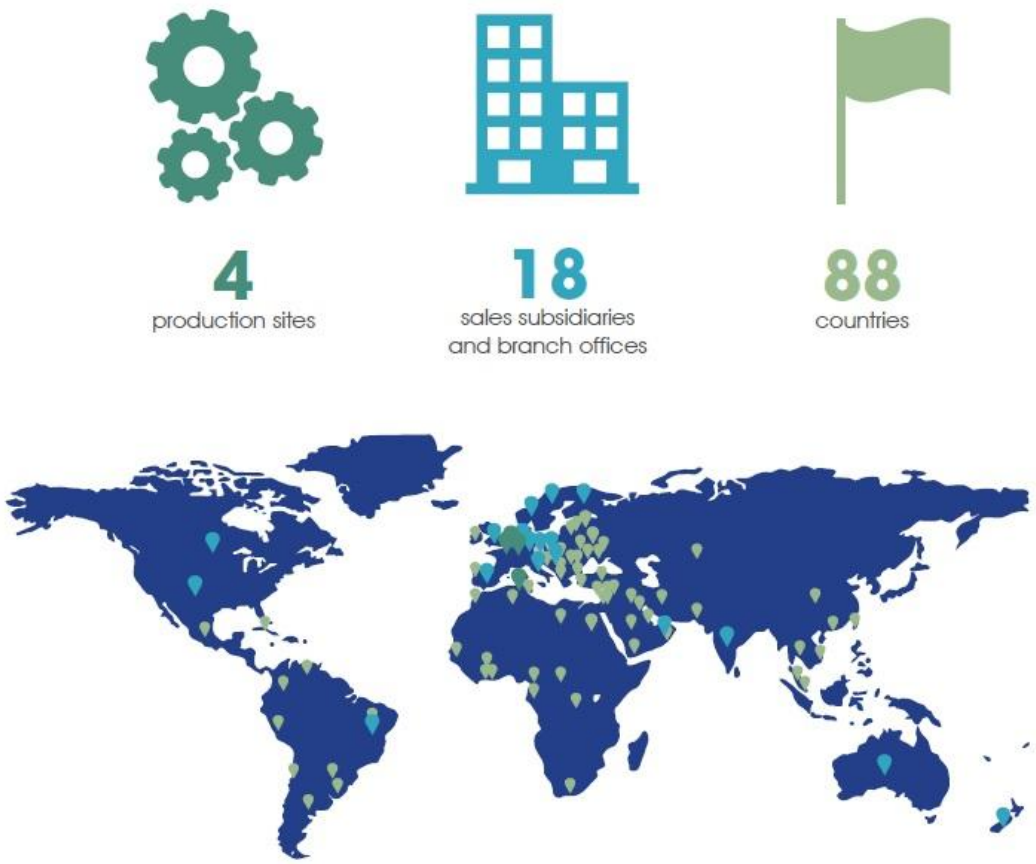

Figure 1 - Macopharma's worldwide presence (Source: [1])

2.1: Macopharma's Business

Throughout the years Macopharma has specialized in many activities associated with health care and blood products and services. The company has excelled and continues to seek advances in each of its core businesses through the work of its four main divisions.

2.1.1: Transfusion

As previously stated The Transfusion Division is responsible for the design and production of devices intended for the collection and preparation of blood components that meet high quality and safety standards.

Figure 2 – Transfusion division (Source: www.macopharma.com)

Adaptable to each need, the blood collection systems represent safe and easy-to-use products for healthcare providers while assuring maximum comfort for blood donors. It consists of blood pack kits, automated blood mixers and tube sealers. The accessories provided with the blood bags in the pack kits prevent accidental needle stick injuries and enable sampling and blood testing, reducing the donation time and the risk of bacterial infection.

Figure 3 – Blood pack kit, the blood mixer Macomix and the Macoseal device

(Source: www.macopharma.com)

For blood processing, Macopharma offers a wide range of devices from leukocytes reduction filters to centrifuges and blood separators. In the blood safety front, innovative pathogen inactivation technologies like the THERAFLEX UV-Platelets and the THERAFLEX MB-Plasma raises the safety margins by mitigating

contamination risks due to residual leukocytes and bacteria or viruses that may have gone undetected during initial screening.

Avoiding the delays and costs pertaining to the development of new screening tests for emerging pathogens and different strains of known pathogens these procedures are considered a proactive approach of preemptive blood product treatment.

Figure 4 – The Macospin centrifuge and a leukocyte reduction filter

(Source: www.macopharma.com)

2.1.2: Hospital

The Hospital Division came to integrate the intravenously injectable solutions line with other products essential to infusion administration in patient treatment. The ample options of ready-to-use infusions include pain management solutions

(containing analgesics or anti-inflammatories), irrigation solutions (sterile water, glycocoll and sodium chloride) and anti-infectious diseases solutions (antibiotics, anti-parasitic or anti-fungal).

In addition to the solutions, several types of containers such as soft PVC and polyolefin bags are also supplied by the department. Needle-free connectors, protective sterile caps and other security and monitoring devices increase staff safety, reduce risk of double dosing and even allow the safe manipulation of hazardous medicaments used in chemotherapy.

Figure 5 – Analgesic solution, safe connectors and polyolefin bag

(Source: www.macopharma.com)

2.1.3: Biotherapy

Macopharma's investments in research and development have enabled the Biotherapy Division to conceive, develop and deliver cutting-edge solutions for each stage of cellular therapy, from collection, processing and treatment of cells to tissue or organ patient transplantation.

In partnership with other leading biotechnology companies and with academic research teams in France, this division has had major break-troughs in the recuperation of stem cells (undifferentiated cells) from cord blood and bone marrow, the transportation, preservation and storage of tissue in low temperatures (cryofreezing) and the extracorporeal photo chemotherapy (ECP) to treat autoimmune disorder and transplant rejection.

Figure 6 – The cord blood separator Macopress and MacogenicG2 for ECP treatment

(Source: www.macopharma.com)

In addition to enhancing the quality of cell collecting and graft preservation, the safe, ergonomic, high-performance medical devices facilitate biochemical clinical studies, essential to further advances in the biotherapy field.

2.1.4: Masks

As one of the five official suppliers of respiratory masks for the French government, Macopharma provides highly efficient products that satisfy the most stringent requirements to manage epidemics and pandemics.

The Mask Division has expanded its portfolio to serve industrial users, business-to-business customers and especially healthcare providers. Its new range of surgical and pediatric masks with double certification delivers improved breathability while meeting both the highest industry standards and the needs of users.

Figure 7 – General public, industrial and pediatric masks

(Source: www.macopharma.com)

2.2: Research and Development in Automation

The creation of cost-effective and high-quality automates meant the assembly of professionals with specialized knowledge in mechanical, electric-electronic and software engineering. Together these professionals compose a team of research and development in automation with the expertise to meet Macopharma's needs and bring innovations into the field of medical devices.

Rather than being an isolated division with a defined range of activities, the Automation department develops projects in constant collaboration with Macopharma's primary sectors, notably the Transfusion, Hospital and Biotherapy divisions. In a sense, the divisions are the department's clients and the department is the divisions' supplier when it comes to automated machines.

Through an open line of communication the aforementioned divisions provide the Automation department with the specification set that serves as project requirements for a new product or simply for a revision of an existing device. These requirements are a result of the exchanges between the divisions and their final clients.

Each division has a direct relationship with its product consumers and profits from this relationship to gather information on user experience and client requests concerning the automated devices offered. The conduct of market research allows the divisions to single out their products' eventual shortcomings or amelioration points. It also allows the identification of upcoming needs in their field.

The Automation department has its own direct contact with target consumers through installation, maintenance and support procedures. While the divisions' dialog with the final clients focuses in the biological and health aspects of the services offered, the Automation department has a more technical approach due to the team's background training. The additional information completes the specification list for new mission. Figure 8 shows a simplified scheme of how the Automation department is placed within Macopharma's structure.

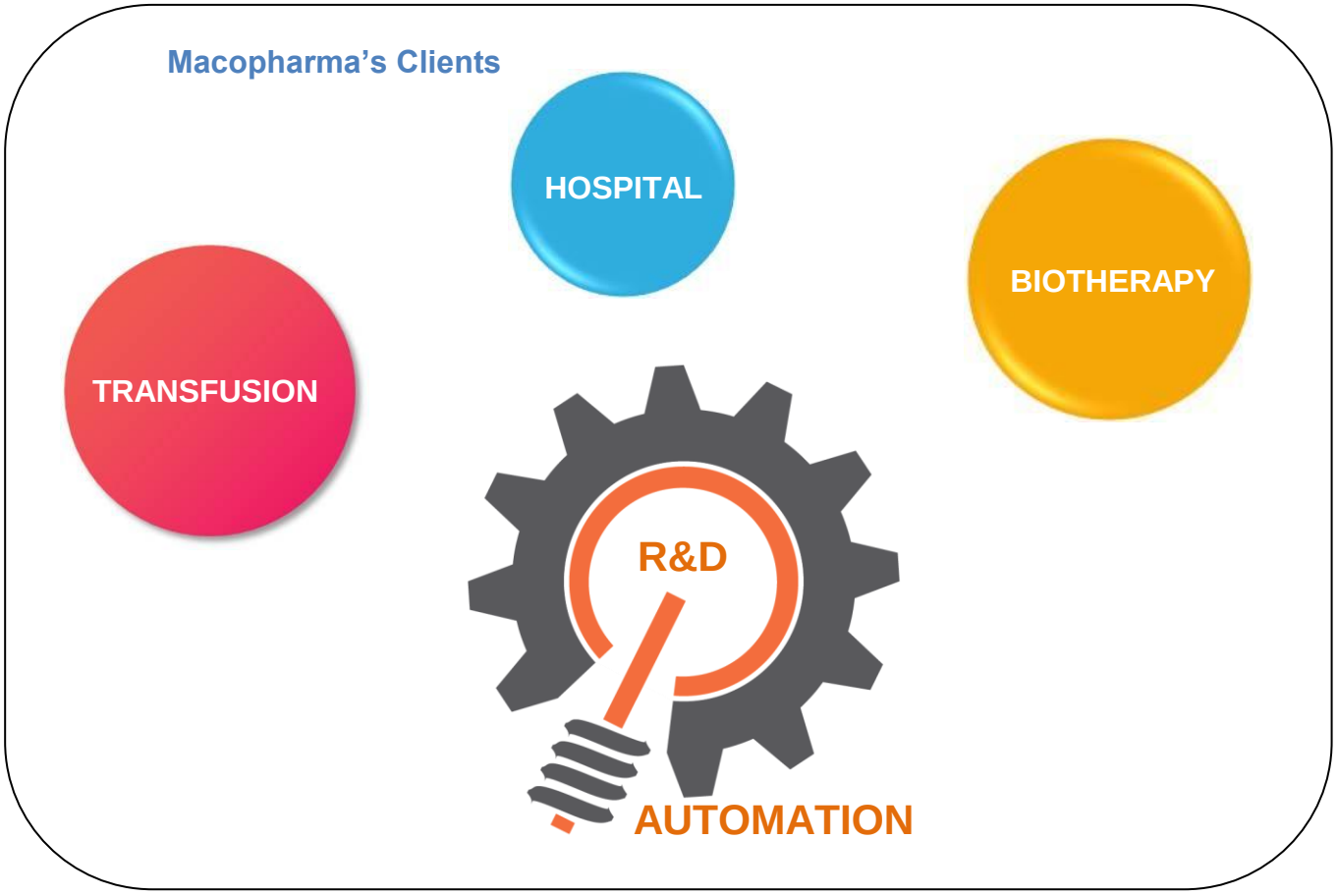

Figure 8 – R&D Automation (Source: own production)

In addition to assembling user requests, the divisions' verifies the regulations pertinent to the application in question. The rules and standards that must be respected in order to ensure the quality of the final product are also documented and passed to the Automation team. The resulting set of requirements is a project's lead-off.

2.2.1: Project Life Cycle and Documentation Process

The life cycle of a project consists of many steps. Each step is generally accompanied by an official document. The documentation process is essential to

ensure Macopharma's quality standards. It also aids in proving project conformity to the applicable regulations and improves overall product maintainability. The diagram shown in Figure 9 summarizes a project's work flow in macro phases.

Aware of the specification set and the regulatory requirements, the development team will analyze the demands and define a list of functions to answer the requests. With these functions in mind, the system can be designed and its architecture determined. The modularization of the system is a well stablished practice, since it helps to organize the work and may even allow for the recycling of modules in other projects. The functionalities are allocated in modules according to their purposes.

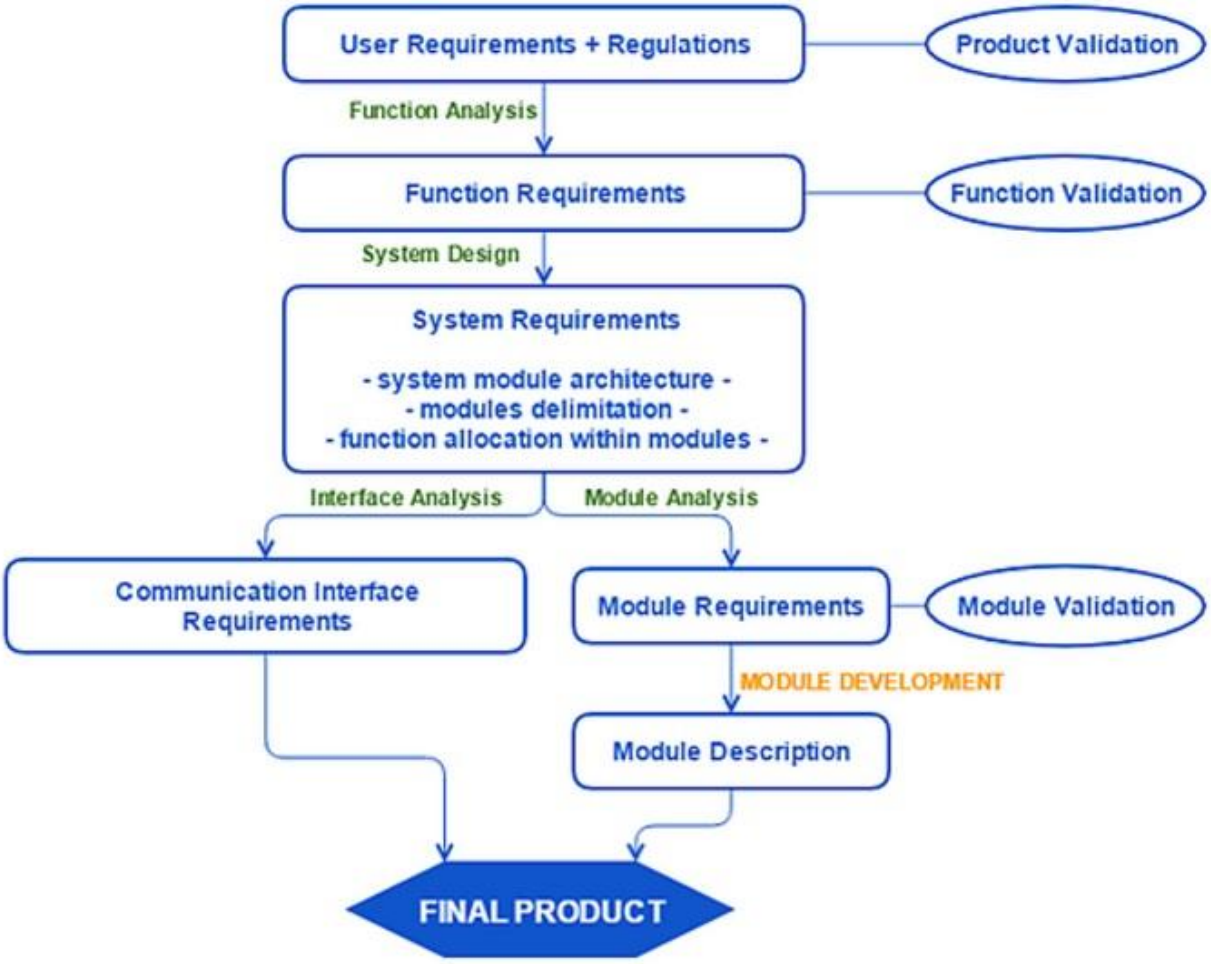

Figure 9 – Project life cycle and work flow (Source: own production)

All modules are then analyzed and a list of detailed requirements is elaborated for each one taking into account all its figuring features. The development and implementation of the modules is strongly founded in these specifications and, as such, it cannot take place without them. Once the modules are implemented, the chosen solutions are described and justified. Throughout the project's life span there are validation stages to verify that the implemented features meet the specified requirements.

In the diagram every transition represents an activity that results in a document containing the information described in the graph's square boxes. The ellipses mark the validation phases where it is ascertained, by a documented test plan, if the developed features have attained their intended goals. In other words each specification document has its pair in a trial description that comprises all tests and results needed to validate the proposed solutions.

Since the object of the present document revolves specially around the implementation stage (highlighted in orange in Figure 9) and focus on software rather than hardware development, it is worth noting that a simple and defined methodology is followed when it comes to this step.

As each module is implemented separately, the software team makes use of Redmine, a project managing tool, to create tasks within the modules scopes and thus better organize and prioritize the work [12]. Deadlines for the tasks' conclusion can be defined based on the project's delivery time and on existing preconditions. This tool allows the team leader to assign the tasks according to personnel availability, experience level and any other criteria deemed of importance. The task's responsible analyses and estimates a duration for its development. The tool also enables commentaries on the tasks' progress and the allocation of man-hours which prove to be useful information for statistics and future estimations.

Redmine calculates a Gantt chart to monitor the project overall advancement and connects with TortoiseSVN [13], a version control tool also used by Macopharma's Automation department. In addition, every Monday the software team gathers to share information and the status on each current project. Together these tools and practices increase product and version security, the level of organization and the informational transit between team members.

Chapter 3. Blood, Blood Products and Services

Blood is the life-maintaining fluid that circulates through the human body, regulating body temperature, transporting nutrients and oxygen to the cells and carrying off carbon dioxide, ammonia and other waste products for filtering in the lungs, kidneys and liver. Other critical functions are forming blood clots to help the cicatrization process and conveying cells and antibodies to fight infection.

A typical 70 kg adult has a total blood volume of approximately 5.5 liters. This volume varies depending on height and weight of a person as well as his/her state of hydration. The blood is composed mainly of erythrocytes (red blood cells), leukocytes (white blood cells), thrombocytes (platelets) and plasma.

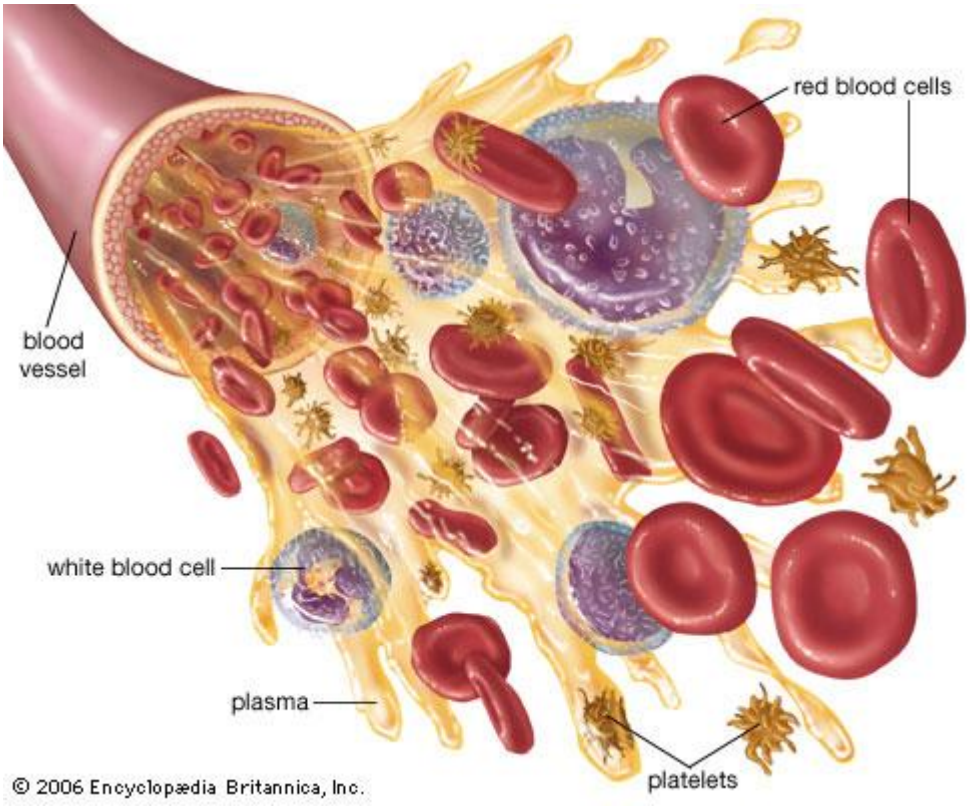

Figure 10 – Blood Components (Source: Google Images)

3.1: Erythrocytes (Red Blood Cells)

Known for their bright red shade that gives the blood its characteristic color, erythrocytes, commonly called red blood cells, are the most abundant cells present in the blood stream, accounting for 40 to 45 percent of its total volume.

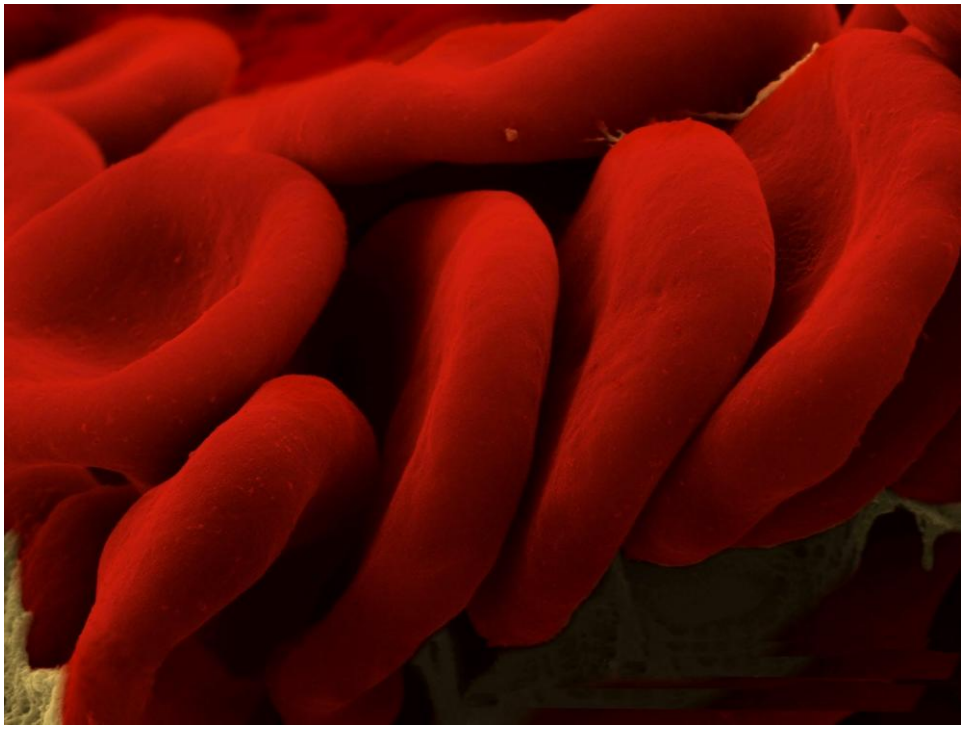

Figure 11 – Microscopic view of Erythrocytes or Red Blood Cells

(Source: Google Images)

Shaped as biconcave disks, these cells have no nucleus which renders flexibility to fit through the various sized blood vessels within the human body. The lack of nucleus also contributes to accommodate maximum space for hemoglobin, a special iron containing protein that binds with oxygen. When the erythrocytes, traveling in the blood, reach the body's tissues and organs this biomolecule releases the oxygen allowing cellular aerobic respiration to take place thus fueling metabolic functions. The red blood cells are produced in the bone marrow and have a life span

of about 100 to 120 days in the circulatory system before their components are recycled by phagocytosis [2].

3.2: Leukocytes (White Blood Cells)

Leukocytes, also called white blood cells, are the body's first and foremost line of defense. They protect against bacterial, viral, parasitic and fungal infections, allergies, tumors and other diseases by identifying and attacking foreign agents. Deriving from a cell produced in the bone marrow that has the potential to differentiate, leukocytes present in many types each one specialized in different immunological functions. All leukocytes, however, have a nucleus, distinguishing them from red blood cells and platelets.

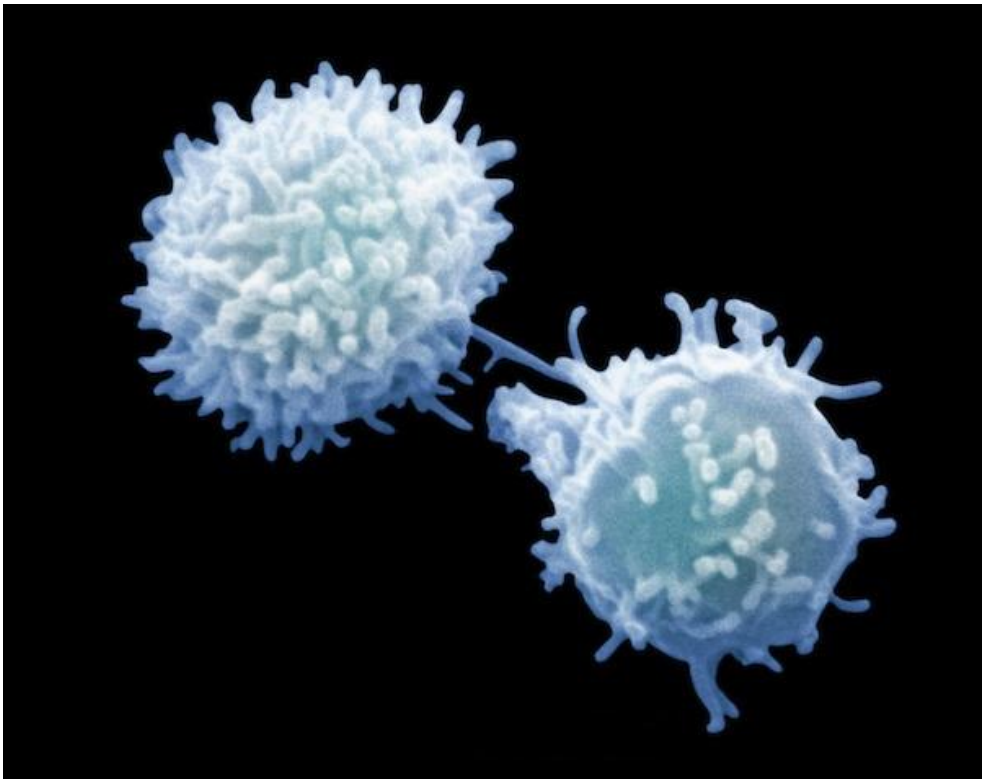

Figure 12 – Microscopic view of Leukocytes or White Blood Cells

(Source: Google Images)

In addition to circulating in the blood, these cells are found in other tissues throughout the body, notably the spleen, liver and lymph nodes. In case of injury or infection they are quickly mobilized to the affected area. They are also responsible for the removal of the dead red cells remains by phagocytosis, as mentioned in the previous section.

Although a reassuring presence in most cases, white blood cells can cause autoimmune disorders (when the body's immune system attacks and destroy healthy cells by mistake) and graft-versus-host disease (when the transplanted tissue's leukocytes recognize the recipient's cell as foreign and tries to neutralize them, causing transplant rejection) [3].

White blood cells represent less than 1 percent of total blood volume and have a short life expectancy of 18 to 36 hours [2]. The number of leukocytes in the blood is often an indicator of disease: an elevated proportion generally suggests infection.

3.3: Thrombocytes (Platelets)

Unlike erythrocyte and leukocytes, thrombocytes, most simply named platelets, are not actually cells but rather small cell fragments. They do not possess nucleus and are about 20 percent smaller than the red blood cells. Along with clotting factors (chemical compounds in constant circulation in the blood), the platelets' main functions are to prevent bleeding and repair damaged blood vessels.

Being lighter than other blood components, they are forced from the center of the blood flow to the walls of the blood vessel. The surface of the walls is lined by endothelium cells that stop any agents from attaching to the duct. If an injury causes a tear or a cut in the endothelium layer, however, a signal is sent and the platelets are activated, changing from their normal plate form to a star or octopus-like shape, with long extended filaments, or tentacles, and adhering to the broken region. They cluster together providing an initial seal and marking the start of blood coagulation.

The result of this process is a fibrous clot which covers the wound and forms a platform upon which new tissue is woven promoting healing.

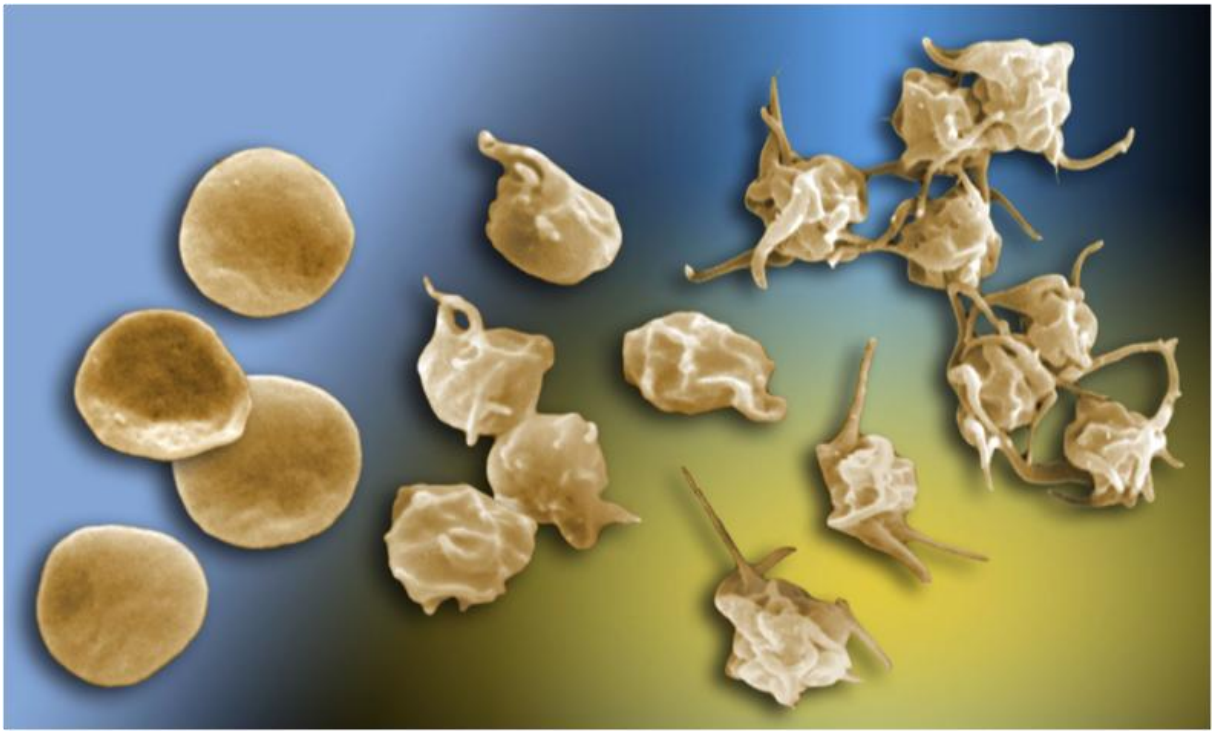

Figure 13 – Thrombocytes or Platelets in their inactive (left) and active (right) forms

(Source: Google Images)

Platelets disorders are well known in the health department. High platelet count can cause spontaneous clots and consequently heart attacks and strokes. Low platelet count is characterized by easy bruising and frequent and/or excessive bleeding.

3.4: Plasma

The liquid component of blood is called plasma: mostly water (92%) with a mixture of sugar, fat, protein and salts. It presents as a yellowish fluid and it constitutes approximately 55 percent of total blood volume. It mainly serves as a

conduit to transport blood cells through the body along with nutrients, waste products, antibodies, clotting compounds, chemical messengers, such as hormones, enzymes and proteins that help maintain the body's fluid balance.

Figure 14 – Plasma bag (Source: Google Images)

3.5: Pathogens and Antigens

The human body is a complex and intricate ecosystem where cells co-exist in symbiosis with bacterial, fungal, and protozoan species. These microbes are part of the normal flora and usually aid in essential body functions (e.g. *lactobacillus* bacteria in the digestive system help the body to absorb nutrients and balance the intestine's pH level). Nevertheless they may cause damage in abnormal conditions, in case of immune deficiency or injury. In addition, humans are always infected with viruses, most of which, however, rarely become symptomatic.

Pathogens are all foreign microorganisms that, in contrast with the normal flora, cause diseases when introduced in the human body. They are bacteria, viruses, protozoa and fungi specialized in infecting a host, avoiding its innate

immunological system, multiplying and spreading to new hosts. It is argued that the responses they induce from their hosts (e.g. cough, diarrhea or bleeding) are part of the strategy to enhance their propagation. Figure 15 shows various types of illness-inflicting pathogens: 1) the Ebola virus; 2) *C. gatti* fungus that causes Cryptococcocal disease; 3) Human Immunodeficiency Virus (HIV) that can lead to Acquired Immune Deficiency Syndrome (AIDS); 4) Dengue virus; 5) *Giardia intestinalis* protozoan parasite that causes Giardiasis; 6) *Streptococcus* bacteria may cause meningitis.

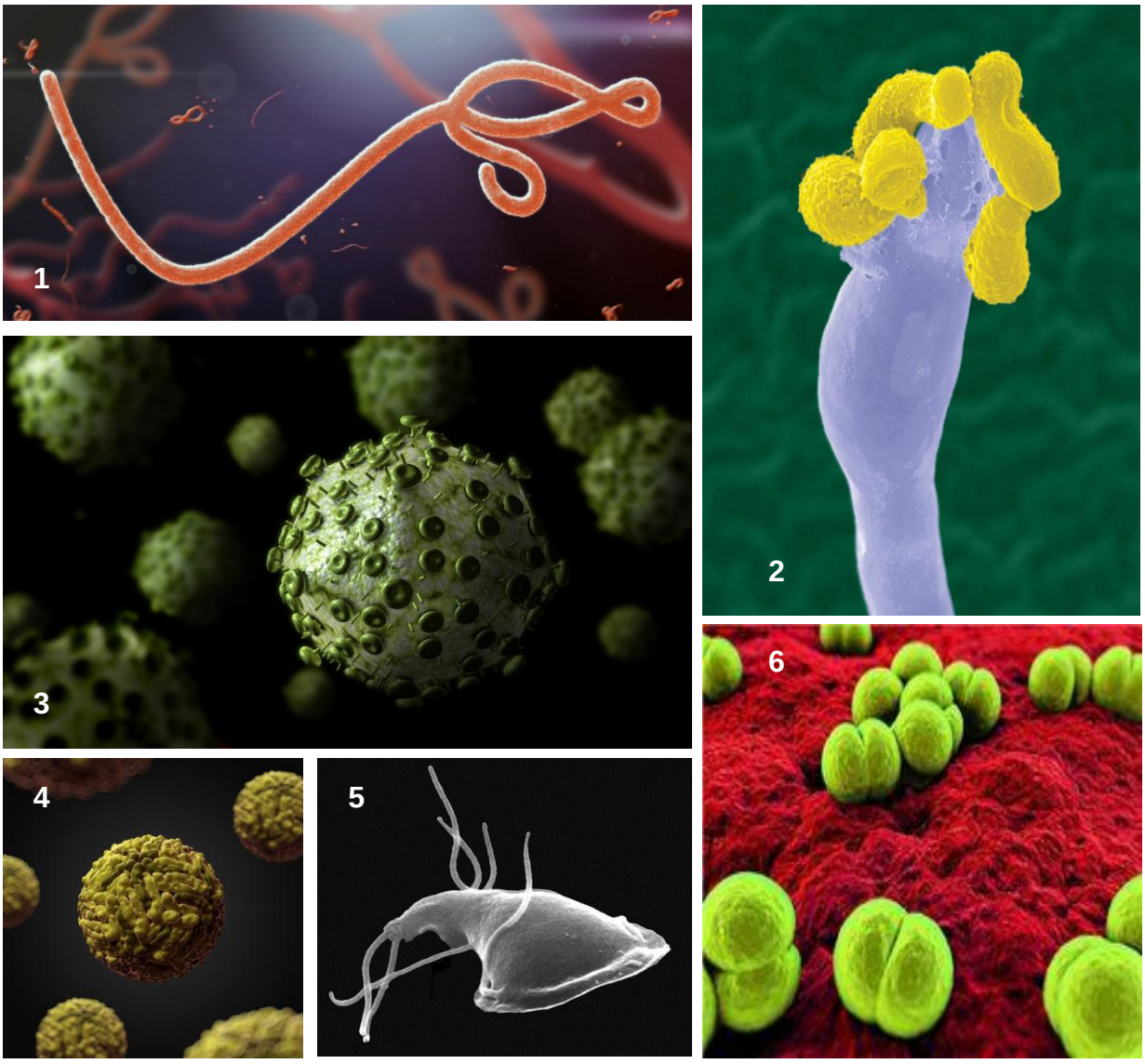

Figure 15 – Microscopic view of various Pathogens

(Source: Google Images)

An antigen is a substance that triggers a response from the immune system. Leukocytes can identify these substances as hostile and foreign, thus setting off chain reactions for the defense mechanisms. They are often found on the surfaces of pathogens.

3.6: Platelet Concentrate Transfusion

Having presented the importance of platelets in the coagulation process, it is easy to understand why platelet transfusion is increasingly being used to treat or prevent bleeding in patients with low platelet counts (due to diseases such as leukemia) or who have suffered massive blood loss (in case of injury, major operation or severe infection).

The platelet concentrate is generally obtained from the buffy coat layer formed when whole blood (i.e. donated blood with all its components) undergoes centrifugation. Due to their distinct densities, the different constituents separate from one another in three visually identifiable phases as shown in Figure 16. The buffy coat, so called for its yellow-brown color, contains most of the white blood cells and platelets.

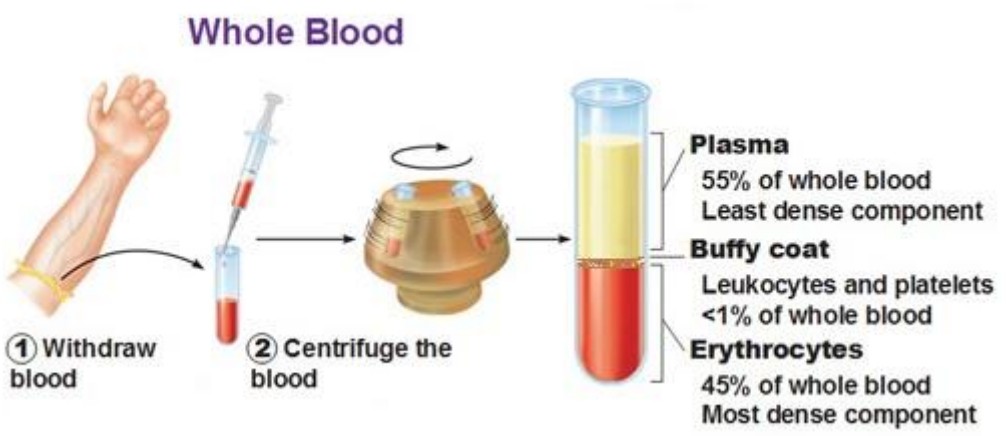

Figure 16 – Blood Separation by Centrifuge (Source: Google Images)

Following the centrifugation, the heterogenic blood bag is transferred to a press machine that physically separates each layer into different bags. Since the buffy coat represents a very small portion of the donated blood, four to six bags need to be pooled together in order to allow for platelet transfusion. These bags generally originate from multiple blood donors. Although there are donation procedures that enable the collection of an acceptable quantity of the buffy coat from one single donor, they generally imply a long collection time (up to 12 hours), which represents a more arduous procedure for the donor. In most cases the pooling process is preferred in view of its simplicity, shorter collection time and lower costs.

As previously stated, the buffy coat has a high concentration of white blood cells as well as platelets. Therefore a leukocyte reduction must be performed. Ultimately the platelet concentrate is the result of passing the contents of a buffy coat bag through a leukocyte-deletion filter.

3.7: The THERAFLEX UV-Platelets Procedure

The main problematic for platelet transfusion is maintaining platelet quality at high levels throughout the preparation and storage of platelet concentrates. In fact, the availability of these concentrates is restricted by their short shelf lives due to two major interconnected issues: pathogens and leukocyte contamination and storage-related deterioration known as platelet storage lesion (PSL).

As with all transplant and transfusion procedures, there is always the risk of contamination by pathogens that may be present in the donated tissue (in this case, the donated blood). Remaining white blood cells can also cause grievous reactions to the transfused patient as explained in Section 3.2:. The pooling process briefly described in the previous section and most commonly used for concentrating platelets increases the risk of bacterial/viral infection as well as leukocyte immune responses since the transfusion bag is obtained from multiple donors.

In parallel, PSL, best defined as harmful changes in platelet structure, can occur between the time of blood collection and the time the platelets are transfused

into a patient, greatly affecting transfusion outcome. The preservation of platelet morphology, composition and function is essential to positive transfusion reactions and a successful treatment. In other words, platelet proprieties that render treatment efficacy cannot be lost in the process of obtaining, preparing and storing the transfusion bags.

Both of these issues are intrinsically related and one cannot be addressed regardless of the other. A balance between reducing pathogen and leukocyte contamination and maintaining platelet quality and function must be considered when trying to produce purer and safer platelet concentrate with extended shelf life. In the last decade, much progress has been made in this front with the development of pathogen inactivation technologies (PITs).

The THERAFLEX UV-Platelets, developed by Macopharma, is an innovative PIT that uses shortwave ultraviolet light to inactivate pathogens and residual white blood cells. Based on the well-established sterilization capacity of the UVC light, for this reason also called germicide light, the procedure causes maximum damage to bacterial and viral DNA/RNA with little-to-none platelet deterioration. This is the result of UVC light application at a wavelength of 254 nm which closely coincides with the highest absorption range by pathogens' nucleic acids (approximately 260 nm) and minimum absorption by platelet proteins [4], as shown in Figure 17.

Optimal delivery and penetration of UVC light is attained when the illumination is combined with the orbital motion of the platelet concentrate bag which leads to uniform exposure.

Although most PITs employ ultraviolet illumination, notably with UVB and UVA lights (wavelength range from 280 to 320 nm and from 320 to 400 nm respectively), the THERAFLEX UV is the only procedure that does not require the addition of a photosensitive agent to target pathogenic DNA/RNA. This singular characteristic represents a great advantage compared to other technologies since it eliminates the need to filter the added photoactive compound in order to prevent chemical contamination and toxicity of the platelet concentrate. It renders simplicity and security to the pathogen inactivation procedure.

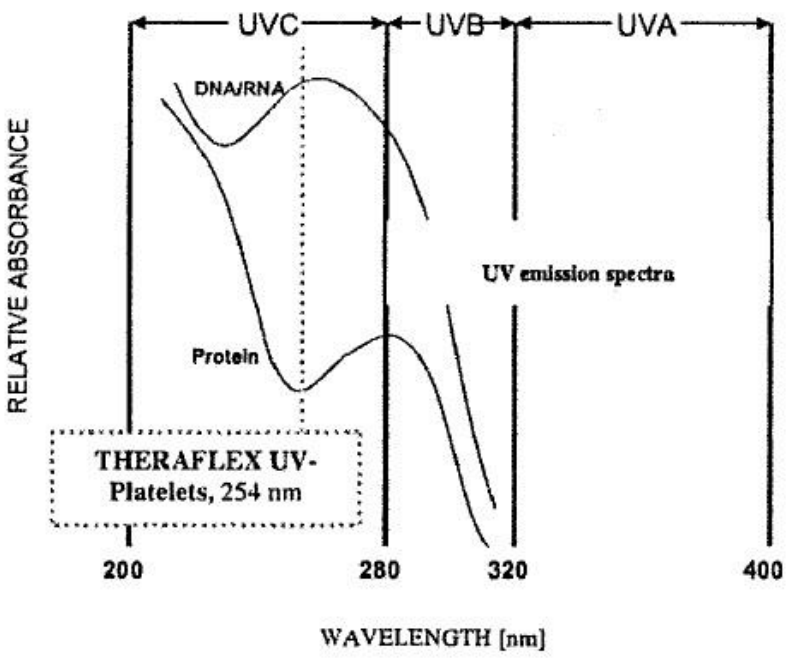

Figure 17 – Absorption Spectrum for Pathogen DNA/RNA and Platelet Proteins

(Source: [4])

Many studies and clinical trials have been conducted to evaluate and validate this PIT. Through these tryouts the THERAFLEX UV-Platelets has proven to be highly effective in inactivating bacteria, viruses and residual white blood cells, while preserving platelet functions and quality [4].

To perform the THERAFLEX UV-Platelets procedure Macopharma has developed the Macotronic UV illumination device shown in Figure 18. The platelet concentrate is transferred into a special new-generation illumination bag, part of the processing kit also provided by the company (Figure 19), and inserted in the machine for treatment. The platelet unit is then agitated and exposed to the UVC light (double- sided illumination) for less than one minute. The procedure is fully controlled and monitored by the device's embedded system. The system communicates with printers to produce reports and identification labels in order to ensure product traceability and correct documentation.

Figure 18 – The Macotronic UV illumination device (Source: [10])

After the completion of the treatment the platelets are transferred to the storage bag of the processing kit and can be released for transfusion with no further processing. In total, the whole operation takes less than 10 minutes, most of that time represented by the two pack transfers, which allows platelets products to be treated and released in the same day [4].

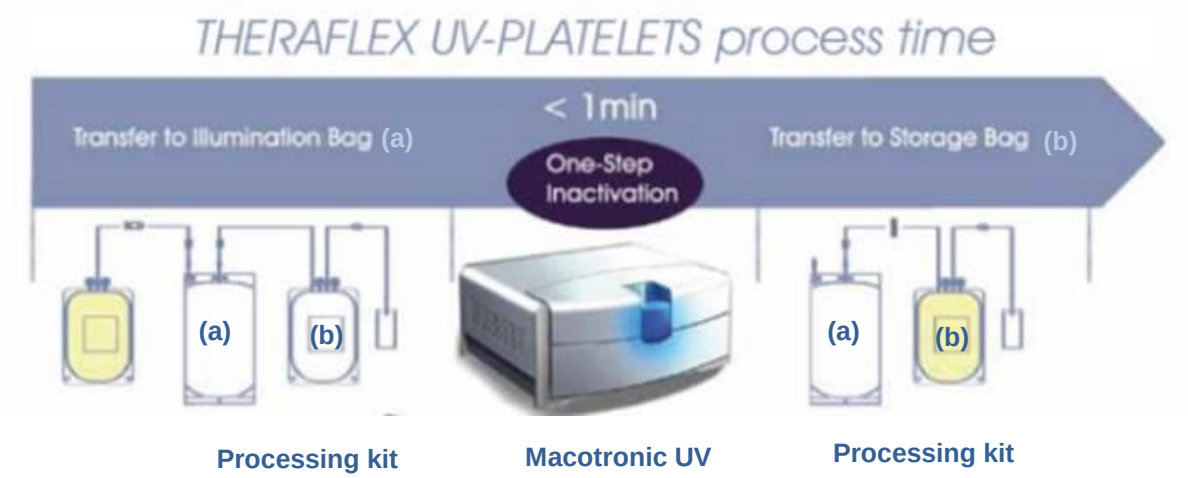

Figure 19 – The THERAFLEX UV-Platelets process

(Source: www.macopharma.com)

3.8: The Macotronic UV

Briefly explained, the Macotronic UV illumination device consists of an aluminum and stainless steel based structure, a LCD touch screen, a front panel door, an ejectable tray to place and agitate the platelet concentrate bag, six UVC lamps, a CPU board and a Rabbit® microcontroller.

The CPU board runs the Human-Machine Interface (HMI) application and the microcontroller operates under a Dynamic C program to control the machine's motors, sensors and relays. Both of these software components exchange information via a communication protocol. Together they compose the device's embedded system. Although such a system already existed for the machine's prototype, a complete revision was performed and a new version was developed and released. The implementation of this new version is the focus of this document and will be further explored in the following chapters.

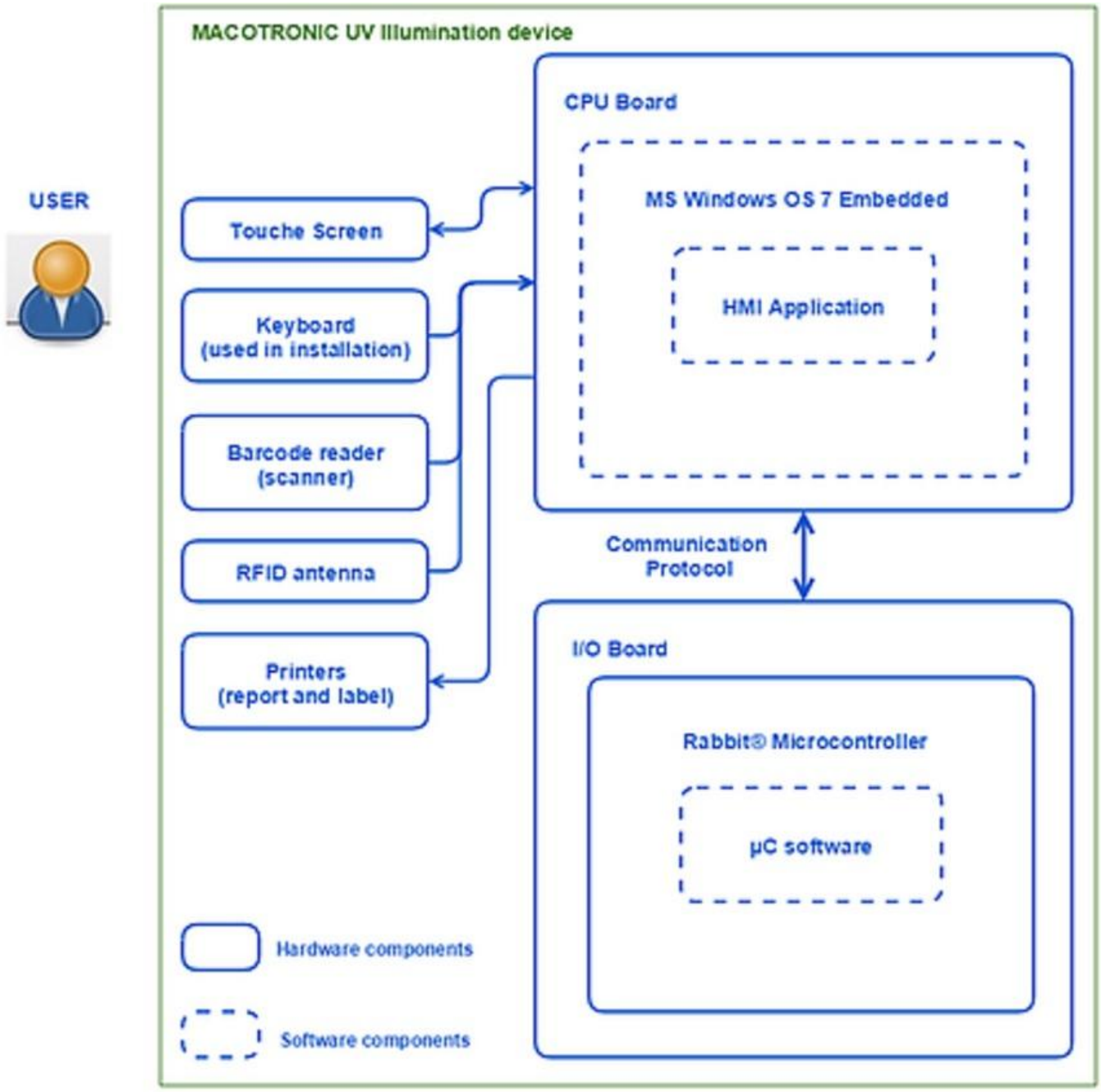

Figure 20 – Macotronic UV's basic structure (Source: own production)

Chapter 4. Macotronic UV: HMI Application

The Macotronic UV's LCD touch screen allows users to interact with its embedded HMI application and, through it, launch and monitor the THERAFLEX UV Platelets procedure. The application also provides users with data management and product traceability.

During clinical trials of the THERAFLEX UV-Platelets procedure the Macotronic UV prototyped version was used by many potential future clients. Since the HMI is a direct link with Macopharma's clients and as such serves as a representation of the company's identity, user feedbacks concerning the application's visual layout was greatly considered in the development of a new graphic design.

Focusing in intuitiveness, the new application is user-friendly and simple to use. Its modern visual aspects also take into account a more standardized design so as to allow users to be familiar with many of Macopharma's blood safety machines.

Originally coded in C#, the HMI software was entirely re-programmed in C++ in view of the language's better performance rates, portability between different operational systems and the general programming facilities it provides (multiple inheritance, pointers, class structure and macros).

Throughout the project, all HMI programming was accomplished with the Qt Creator Integrated Development Environment (IDE) [14]. This IDE offers tools that facilitate the application development. Some of these tools were an exact match to the project needs concerning graphic, flexibility and robustness requirements.

Qt Designer [15], for instance, is a tool that aids in constructing Graphic User Interfaces (GUIs) by providing a visual editor where the developer can compose and customize widgets and dialogs. Another example is the Qt Linguist tool [16]: an excellent support for translating applications into local languages, an aspect of great importance since Macopharma's client network expands internationally. Furthermore, the qMake tool [17] simplifies the building process across different target platforms, which will allow future development and deployment of mobile operating systems such as Android and iOS.

This chapter explores the implementation of the HMI software which runs in the Central Processing Unit (CPU) under the MS Windows 7 Embedded Operating System (OS) its new graphic design, essential inlaid features and added functionalities such as the user management and machine configuration tools. The theoretical basis for the development will also be briefly reviewed.

4.1: Model/View Programming

To better organize the HMI application and increase modularity and maintainability, the model-view-controller (MVC) architectural pattern was adopted. Widely favored when designing user interfaces, this structure divides the software into three interconnected parts [6], as shown in Figure 21, separating the representation of internal information from the way that information is displayed to the user.

The Model layer represents the knowledge of the application. It implements the business logic. It is responsible for storing or retrieving information from an external data source; for converting it into meaningful concepts to the application; for processing, validating and for any other tasks related to data handling. In other words, it defines the application's behavior.

The View layer is the screen presentation of the modeled data. It receives treated information from the Model layer and displays it to the user in customized ways. It acts basically as a presentation filter, highlighting certain attributes while suppressing others. It does not own and therefore cannot alter the core data since that is the Model's domain. A view item needs to be associated with a model in order to properly function, to show the appropriated information.

The Controller layer defines the way the interface reacts to user input. With the aid of both Model and View layers it handles user requests rendering an appropriate response either modifying a model's state or updating a view. In doing so it represents the link between user and application.

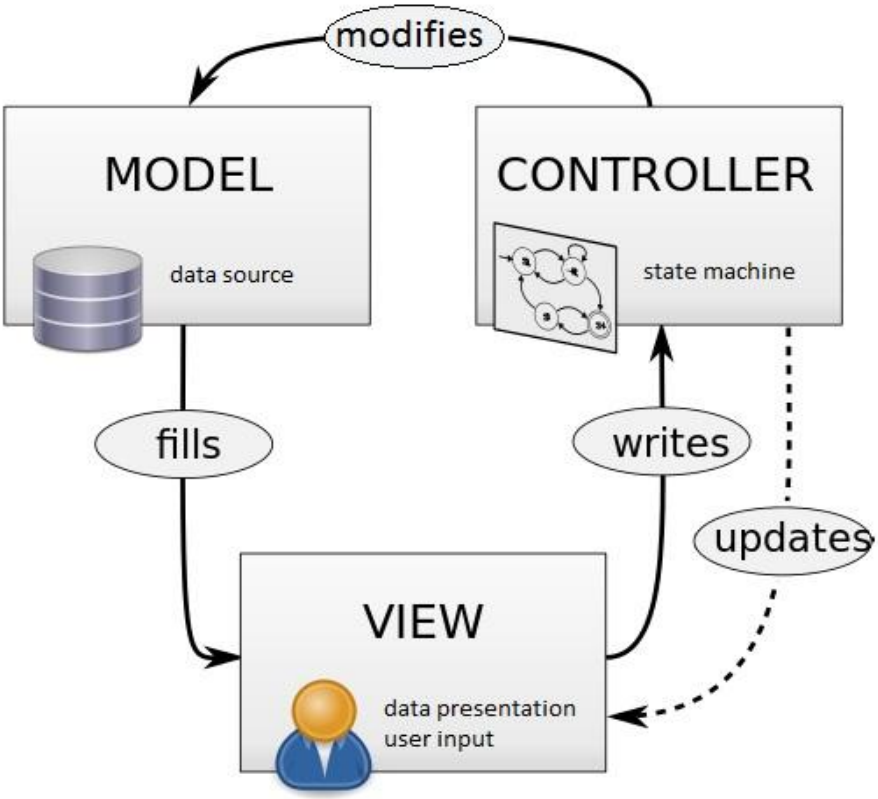

Figure 21 – Simplified diagram of model-view-controller architecture

(Source: Google Images)

Slightly adapting the MVC pattern to simplify the structure while maintaining its principles, the Qt framework combines view and controller objects. This junction means that, in addition to managing overall layout of data presentation, View classes have embedded controllers to handle item navigation, selection and edition. They can also make calls to Model classes to launch a desired behavior. Qt provides abstract classes for the Model and View layers that can be subclassed in order to create specialized components and attain the full set of functionalities needed for each application [5]. These objects communicate using the signal and slot mechanism, a central feature of Qt which is explained in the following section.

4.1.1: Signals and Slots

The signal and slot mechanism can be perceived as a cause and effect reaction. When an object suffers a change in its state that may be of interest to other objects it can be said that an event occurred. In light of this event, a signal is emitted. A slot is a function that is called in response to a specific signal to which it is connected. Hence, a slot can be considered the effect of the cause represented by the signal. For example, if a Close button is pressed in a dialog window (cause), a *close*() function may be called to perform the action of closing the dialog (effect).

One signal can be connected to as many slots as needed and vice-versa. A signal can also be connected to another signal, implying that when the first is received, the second will be immediately emitted. Nevertheless, signals and slots are loosely coupled. In other words, the object that emits a signal is not aware if a recipient exists and/or which slot or slots are connected to the signal. In the same way, a slot does not require knowing if it is connected to any signals or to which signals in particular. It can be used as a normal function. This system enables data encapsulation and ensures that truly independent software components can be created with Qt.

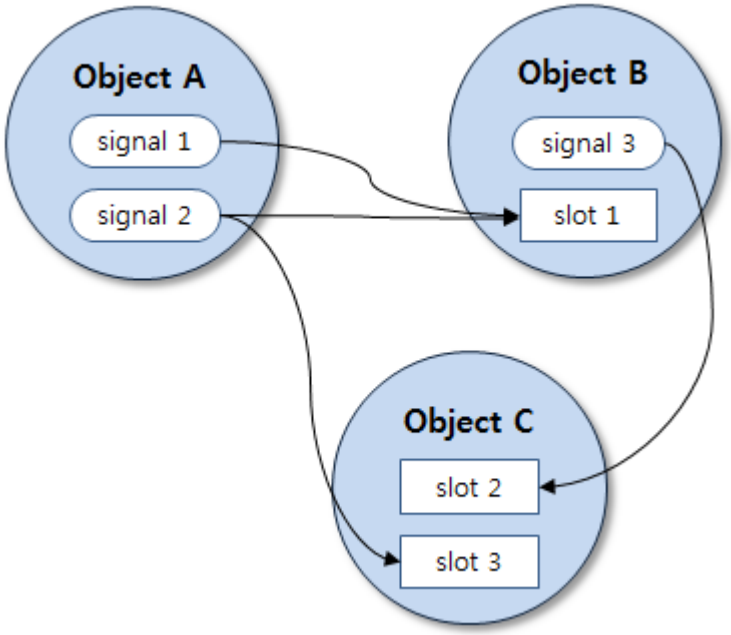

Figure 22 – Signal and Slot connections (Source: Google Images)

When a signal is emitted the slots connected to it will be executed immediately and independently of any GUI event loop, as would any regular function. The execution of various slots connected to the same signal follows the order of the connections: the first slot connected will be the first executed. The tasks programed after the emission of the signal will occur once all slots have returned. Queued connections can be used to alter this behavior so as to carry out the tasks following the emission of the signal before the execution of the connected slots.

Signals and slots can take any number of arguments of any type. However, the signature of a signal must match the signature of the connected slot to ensure type safety. In fact, a slot may have a shorter signature than the signal it receives because it can ignore extra arguments, but the opposite cannot occur. Since the signatures are compatible, the compiler can help detect type mismatches when using the function's pointer-based syntax [5].

As previously stated slots are regarded as normal functions. Thus they can be called directly by any object with the right access level. They follow the regular C++ rules and can be defined as virtual, which can be useful when subclassing widgets. When considered in a signal-slot connection, however, the slot may be invoked by any component regardless of the access level. This means that a signal emitted from an instance of an arbitrary class can cause a private slot to be invoked in an instance of an unrelated class.

Qt offers a variety of default widgets with predefined signals and slots, but it is common practice to subclass these widgets, customizing signals and slots to serve the application's needs. Indeed, the Macotronic UV application was founded on subclassing the base user interface class *QWidget*.

4.2: Graphic Design

This section focuses in the graphic design of the application or its View layer. The user interface is a composition of screens. Every screen corresponds to a top-level item of the View layer and thus is a class always named with the suffix "View".

For instance, the class associated with the screen that appears at the machine's start-up is called *InitializationView*. All the views derive from a superclass *View* that in turn inherits from Qt's *QWidget* class. This class is the bedrock for the View layer since it receives mouse, keyboard and other events from the window system and paints a representation of itself on the screen.

The window system is the platform onto which the GUI's composition is build. Inheriting from *QMainWindow* (also a subclass of *QWidget*) the *MainView* class represents the founding visual element of the HMI. It is the primary widget and, as such, it is not embedded in a parent widget but rather serves as a parent for all other views. In other words, it is a canvas for the application's screens to be portrayed in. Figure 23 shows a simplified relational diagram between the aforementioned classes.

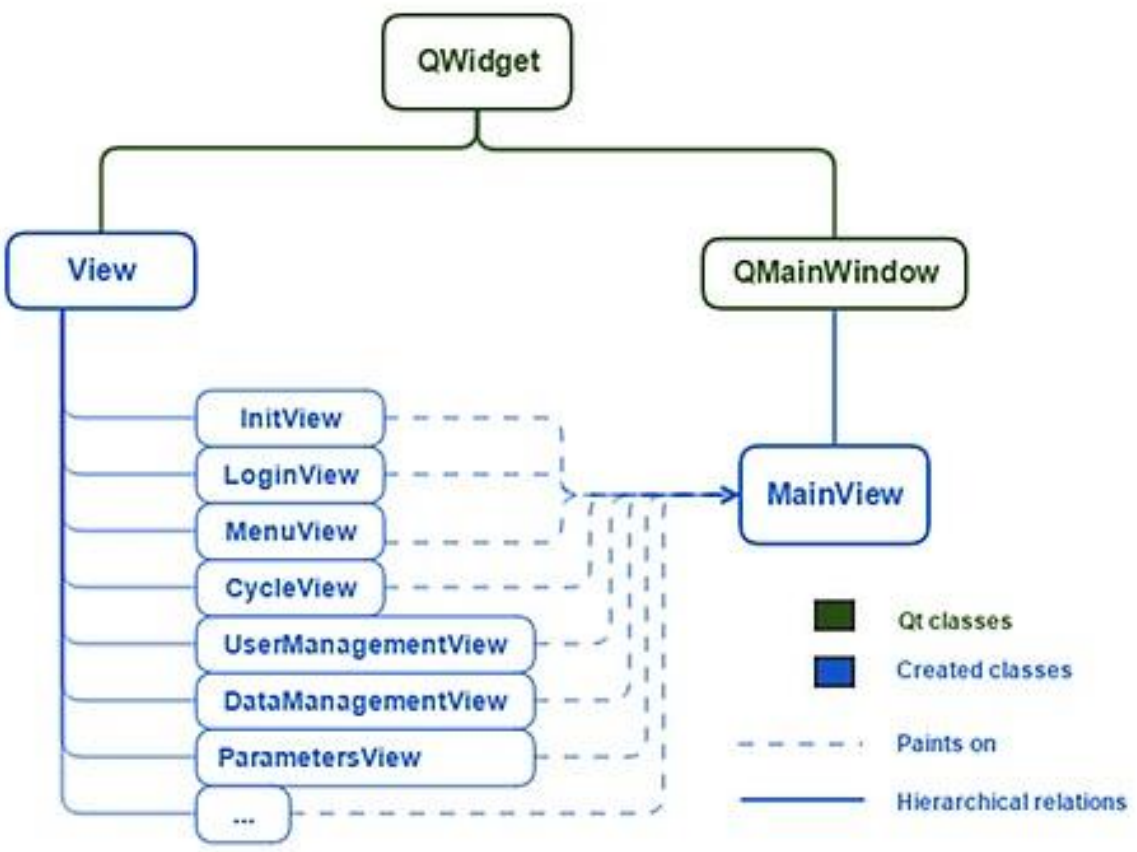

Figure 23 – View Layer (Source: own production)

Designed with Qt Creator's visual editor, this elementary component is divided in three parts: a central area where most of the information is displayed; a horizontal bar at the bottom where action buttons are arranged according to the application's needs; and a vertical bar at the right side where other interactive and informational items can be placed. Figure 24 shows the main view (in red) in its "undressed" form with its basic components highlighted (the bottom widget in green and the right widget in black).

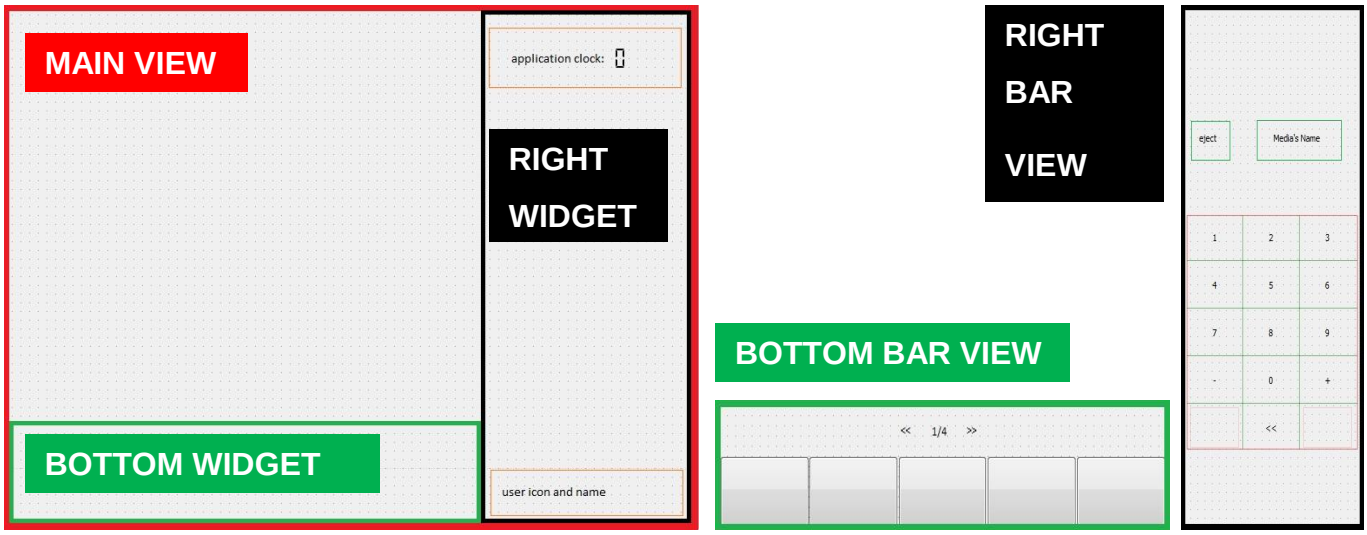

Figure 24 – Main View design (left); Bottom and Right Bars Views (right)

(Source: own production)

The encapsulation of widgets adds flexibility and allows the use of multiple views at the same time. In this case, three views can be used simultaneously, each in a specific area of the main view. This does not mean, however, that three views must always compose the main window. The *BottomBarView* and the *RightBarView,* generic widgets presented in Figure 24, are combined with different bigger views (placed in the central area). They are programmed to provide certain basic features needed in various stages of the application. The *BottomBarView* integrates buttons to trigger specific actions and the *RightBarView* assembles a virtual numeric keyboard (among other minor services).

As each application state has its particular set of actions to perform, its view can make use (or not) of one or both the aforementioned views. Figure 25 illustrates an example: during the login of a user by password, the *RightBarView*'s virtual keyboard is used as an input tool and appears over the main view's background image (picture in light blue). On the other hand, if the user identification is via a scanned barcode the *RightBarView* is not required and consequently does not appear, leaving only the background image at the right side of the window.

The modularization of these objects enables the repeated use of the same views when identical, or similar, services are demanded in different steps of the operation. To manage their behavior, as explained in Section 4.1:, they are associated with the Model layer having each a model to communicate to: *BottomBarModel* and *RightBarModel* respectively. Their generic programming anticipates the need for customization of the functionalities they offer. For instance, the virtual keyboard previously referenced can be reused in different screens with different character options than that of the *LoginView*. The buttons provided by the *BottomBarView* may vary in visual aspects according to the currently required services. It can be seen in Figure 25 that both images contain a button in the middle of the bottom bar, but the icon changes to specify the action that will be trigger if the button is clicked (a barcode scan on the first image and a password entry on the second to represent a change between the methods of user identification).

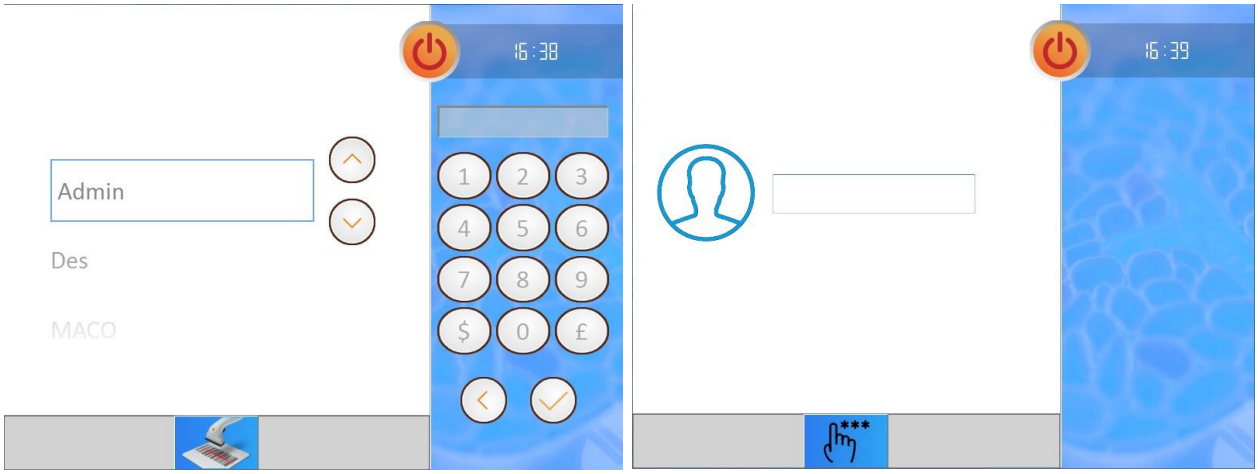

Figure 25 – Login Views: by password (left) and barcode scan (right)

(Source: own production)

It can also be observed that although the *BottomBarView* offers multiple buttons (as shown in Figure 24) only one is needed for the user identification views. Employing a specific number of *BottomBarView's* widgets is another way its services can be customized. Beside their quantity, the position of the buttons can likewise be chosen. Figure 26 presents another screen shot, where the Add, Edit, Delete and Save buttons are depicted.

Regardless of the customization, this design organizes the screens so that the user will always know to find the right tools in the right places: the action buttons will always be located in the bottom of the screen no matter their role and appearance while the keyboard will always be set in the right side no matter what characters it shows. Increased coherence and constancy was a point made by client feedback to improve the overall simple-to-use quality of the application when compared to the preceding software.

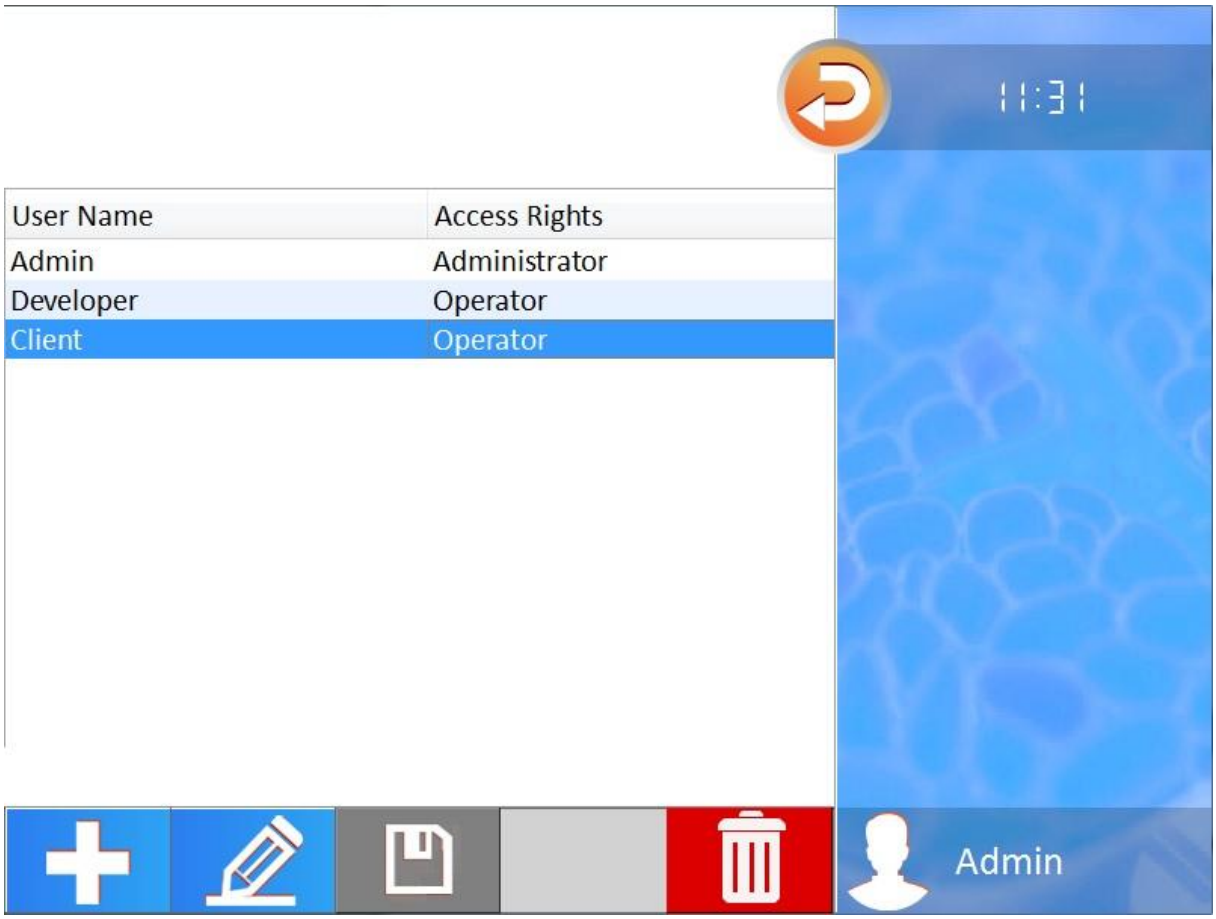

Figure 26 – User Management View (Source: own production)

Another important element from the View Layer is the *GenericPopupView*. It is meant to draw the user's attention to an important message by jumping or "popping-up" over the main view. Much like the *BottomBarView* and the *RightBarView* and as it can be inferred by the class's name, it is recycled to serve in many situations, notably to show error warnings of various types. Even though the information communicated may change, the presentation remains basically the same: a translucent caution themed band superimposing the current screen leaving it with a blurred effect (Figure 27). It can also provide the Yes, No and Ok buttons depending on the problem and if a solution is offered for user validation.

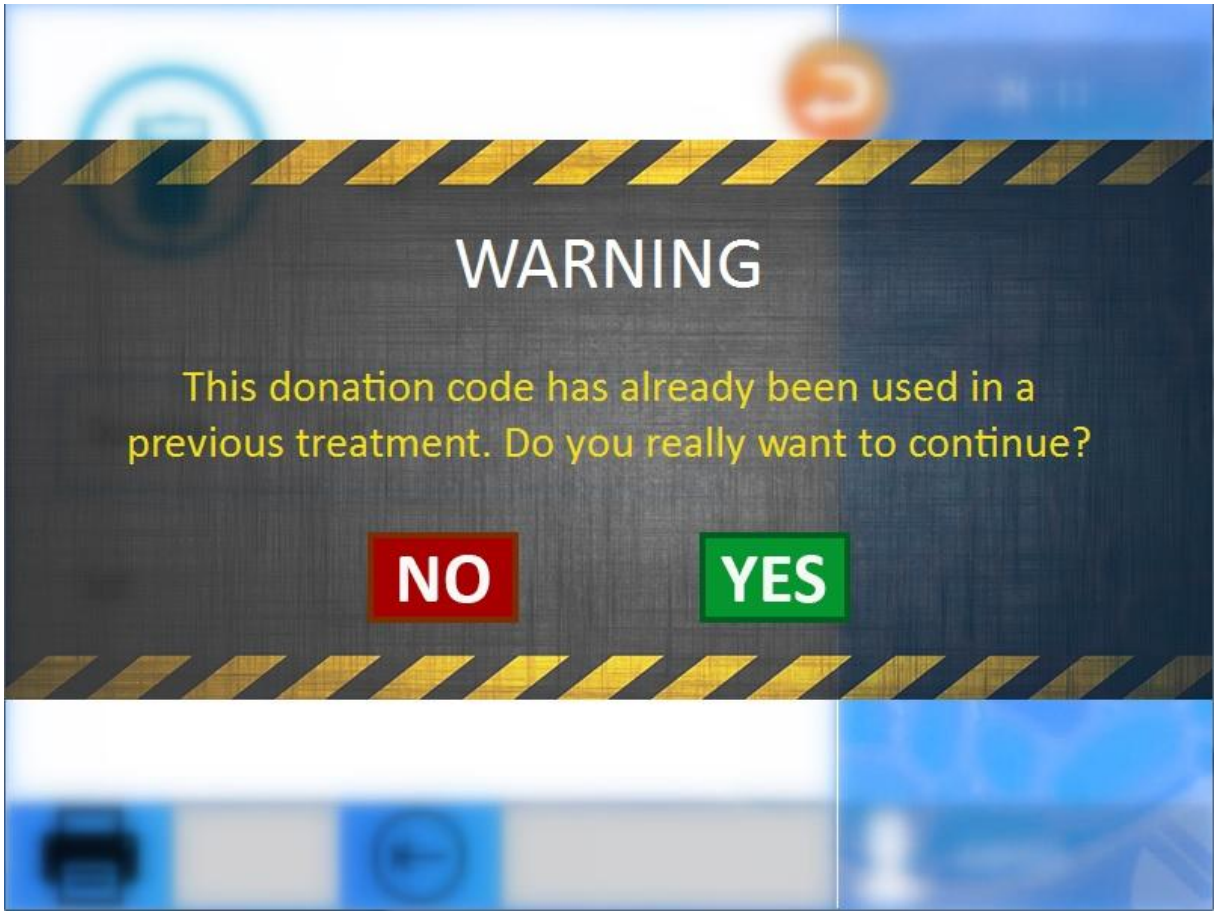

Figure 27 – Generic Popup View (Source: own production)

The imagery – background pictures, icons, colors – selection was as much a part of the HMI's development as the views' programming. Some items were recovered from other recent projects to attempt at a standardization of the blood safety devices. The idea is that clients who acquire several different blood treatment

machines will be familiarized with the embedded applications if the devices share a similar graphic design, easing the clients' adaptation process. Many items however were specially chosen for the Macotronic UV's user interface and are, for the moment, only present in this project. The overall goal of the imagery selection was giving the application a simple, but modern look, easy to comprehend and operate.

4.3: Application Modules

Looking to the future, Macopharma's Automation department envisions many home-made software releases for its current and forthcoming machines. Considering that all the company devices have not only unique features but common features as well – such as user identification process, data management and machine configuration – the concept of a generic platform with recyclable non-specific modules is greatly valued.

Favoring this line of thought the HMI software was conceived in a modular structure. Without rendering operational details for each feature, Figure 28 shows the application's major modules. Through the image, it is evident that there are many modules that could be reused for other devices, providing the same services only with different core information.

With this organizational architecture in mind, the common modules were programmed seeking generalization so that the codes could eventually come to integrate a universal base from where future devices would retrieve certain fundamental features to compose their application. However, other than anticipating the possibility, this project has not further participated in the realization of a universal platform.

To provide its main function each module is composed of both Model and View layer elements. Every view item is connected to at least one model to obtain the information it needs to present on the screen. More often than not, a view communicates with multiple models, since complex features require data from many different sources. As an example, the simple module that allows users to reprint a

report or a product label concerning an illumination cycle comprises a view associated with several models (Figure 29). It must not be overlooked that the module also employs the *BottomBarView* and the *GenericPopupView*, along with their respective models to make use of their services as described in Section 4.2:.

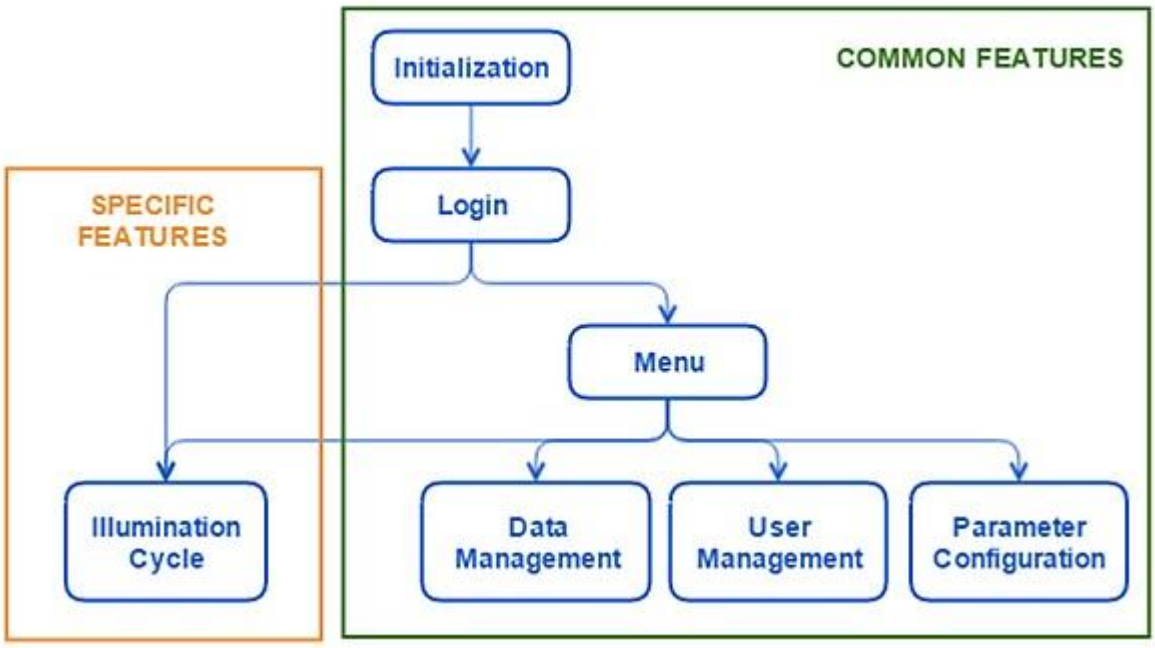

Figure 28 – HMI's operational modules (Source: own production)

Every time an illumination cycle is executed its data is stored in a history file. It is based on this data that the reports and product labels are generated. The CycleHistoryModel manages all existing history files and is needed to perform a search given the intended cycle's bag code. Once the appropriate cycle file is found, the user can select it and press on the Report or Label button from the ReprintView (Figure 30 on the left) – provided by the BottomBarView – choosing to print the associated report or label respectively.

Receiving the user's choice as an input, the ReprintView will convey a command to the *ReportModel* or the *LabelModel* that, in turn, will compile and format the pertinent information from the history file into a report or a label and will subsequently sent it to the specialized printers connected to the Macotronic UV. If there are no files corresponding to the entered bag code, a warning message

appears to the user as it can be seen on the right in Figure 30 (service from the *GenericPopupView*).

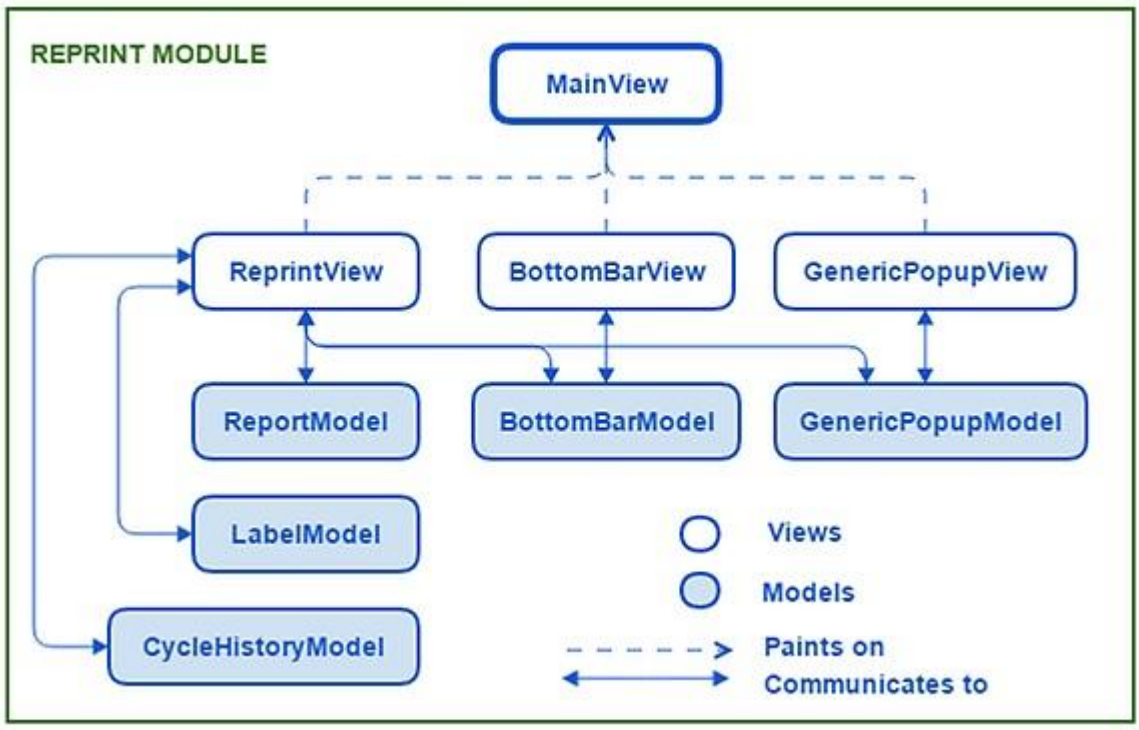

Figure 29 – Reprint Module components (Source: own production)

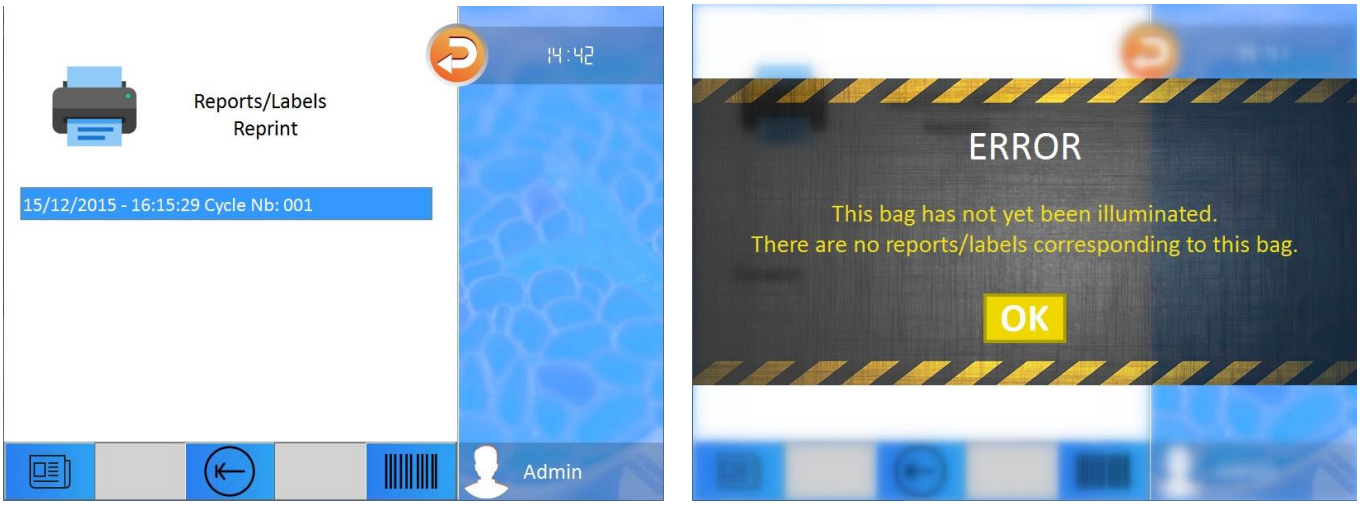

Figure 30 – Reprint View (left) and warning message (right)

(Source: own production)

Other modules utilize the above referenced models as well. Models are islands of specific information that can be accessed by different agents in different moments of time, avoiding code repetition and data duplication. Since they monopolize and concentrate all information belonging to their particular functions, when that information is modified by an operation there is no risk of coherence or synchronization problems: all other modules are sure to receive the correct updated data.

The Reprint module is but a sample used in this document to illustrate the dynamics between the Model and View layers and corroborate the presumed advantages of this pattern. As it provides unpretentious functionalities (printing arranged information) it was not included in Figure 28's diagram of the HMI's main features. The following sections explore a few of the modules' development focusing on some worth-to-mention implementation details.

4.3.1: User Management Module

As already remarked, the Macotronic UV illumination device is protected by a user identification system which is managed by the HMI. When a client acquires one of these machines it is necessary to register a list of authorized users: login names, passwords, badge code and access level. Once this list exists, any of its members can log in using the appropriate entry information.

While the previous version of the application did verify and validate a login before admitting a user to operate the device, it lacked a feature to add, edit and exclude users from this list. Macopharma's technicians had to register all approved users during the installation of the machine via an external tool. It was a procedure both ponderous and prone to misspells. Outside the application the technician could unknowingly misplace or disarrange configuration and/or dynamic-link library files (DLLs) causing other and more serious problems.

Eliminating the need to exit the application to access an external tool, the new HMI, integrates a user management module thus facilitating the task for

Macopharma's technicians. This module also allows the direct handling of the access list by an Administrator-level client user.

The *UserManagementView* presents the registration list in the form of a two column table (Figure 26). The first column contains the user name and the second the access right (either Administrator or Operator). The Add, Edit and Delete buttons enable the administration of the list. When an existing user is selected and the Edit button, circled in green in Figure 31, is pressed the highlighted line will extend to show four editing fields and an option box in the following order: login name, password, password confirmation, barcode ID and access level. This expanding tree effect, shown in Figure 31, was chosen to display user information after careful consideration.

Impossible to ignore, the screen's small size demanded a study of how to best exhibit the editing fields in a clean and effective way. Once again, client feedback was taken into account. The screen's tactile and overall esthetic characteristics when scrolling horizontally where deemed inferior to when scrolling vertically. Also, seeing that, when distended, the selected line will "push" all other lines the need to scroll at all (in this case vertically) will be less frequent to say the least.

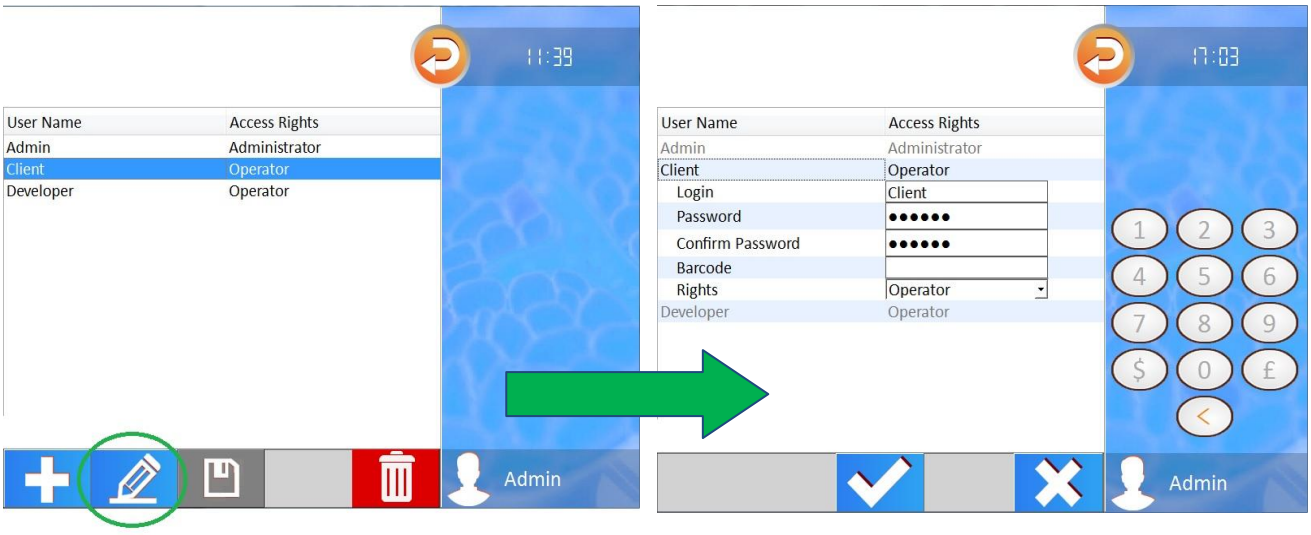

Figure 31 – User Management View in editing mode

(Source: own production)

Other modules could come to profit from this design. Bearing that in mind, the mother classes *GenericTreeView* and *GenericTreeModel* were created to provide, as their names specify, the generic services of listing items in a table-like manner with built-in tree behavior and of handling those items via action buttons from the *BottomBarView*. The *UserManagementView* and the *UserManagementModel* inherit from these super-classes and connect with the *LoginModel* to recover the user information needed to fill in this table-and-tree fashioned container. Figure 32's relational diagram portray the major components of the User Management module.

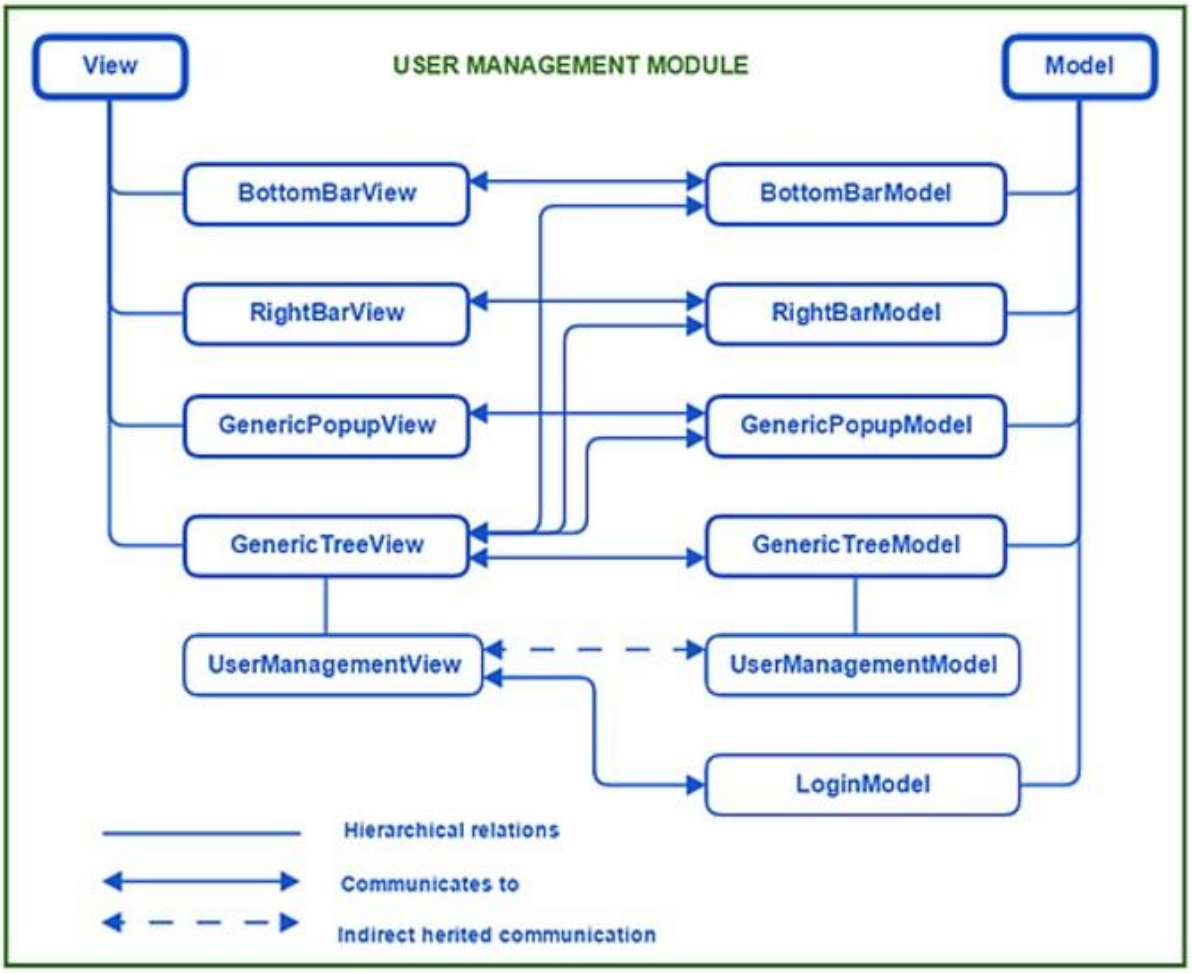

Figure 32 – User Management module (Source: own production)

The *GenericTreeView* class has two constructors: one that requires no arguments and another that demands the name of a model and a view (Table 1 - line 1). The view name builds the current view in the View constructor (Table 1 - line 2),

while the model name serves to find the appropriated model object (Table 1 - line 4). Therefore, every tree view subclass is constructed using its own name and the name of its associated model in these roles, as exemplified by the *UserManagementView* illustrated in Table 2. This allows for the communication between the view and the model indirectly – represented by the dotted line in Figure 32 – and eliminates the need to declare another model as a member of the view class.

GenericTreeView definition	
001	`GenericTreeView(string modelName, string viewName) :`
002	`View(viewName)`
003	`{`
004	`a_pGenericTreeModel = a_pModelFactory->getModel(modelName);`
005	`…`
006	`}`

Table 1 – GenericTreeView class definition

UserManagementView definition	
001	`UserManagementView() :`
002	`GenericTreeView (“UserManagementModel”, “UserManagementView”)`
003	`{`
004	`a_pLoginModel = a_pModelFactory->getModel(“LoginModel”);`
005	`…`
006	`}`

Table 2 – UserManagementView class definition

As a daughter class of *GenericTreeView*, the *UserManagementView* can use its pointer attribute a_*pGenericTreeModel* to capitalize on all the *GenericTreeModel*'s default programmed services. The same attribute gives access to any customized jobs performed by the *UserManagementModel*.

In addition, as formerly showed in Figure 32, the *UserManagementView* communicates with the *LoginModel*. This model contains all information regarding the registered users list. It is used in the User Identification module (or simply Login module as it is named in Figure 28) by the *LoginView* to retrieve the necessary data

to show the list of registered users (if the login method is set to password verification) and to validate either the entered password or the scanned badge code (if the method is set to barcode identification).

In the same way, the *UserManagementView* requests and receives information from this model to display the list on the screen. However, while the Login module only analyses that information, comparing to user input, to grant access to the machine, the User Management module can, through the *LoginModel*, modify the login data source following user actions. The data source from which/to which the *LoginModel* reads/writes user information is an Extensible Markup Language (XML) file. This type of structuration allows for easy, swift and reliable data processing.

Resuming the editing example previously described, after modifying the selected user information and pressing the Confirm button (the one with a check icon showed in Figure 31) the Save button will be enabled and appear colored (when before, as disabled, it was gray). Up until this step the changes made are temporary. Pressing the Save button will trigger a function from the *LoginModel* that will modify permanently the information on the data source.

A view that calls model methods to modify its core directly is the (desirable) result of Qt's strategy of embedding the Controller layer of the MVC pattern in view objects. The implementation of the User Management module confirms the advantages arisen from the adaptation of the original architecture with regard to programming simplicity, as it was discussed in Section 4.1:.

Similar modules were developed to manage barcode formats and configuration parameters.

4.3.2: Mask Management Module

Today, most health provider centers have their own identification and labeling systems which means that different code formats must be registered so that each

client may choose an appropriate pattern for the machine to expect as a platelet bag ID key. The Mask Management module handles the list of possible barcode formats otherwise called masks.

A mask is a mold that serves as a comparison base to verify if the scanned code respects a certain set rules. For instance, a fictional French blood bank may have a labeling system that generates four digits codes preceded and succeeded by the capital letter A for platelet bags that have not yet undergone the THERAFLEX-UV procedure. A sample bag code would be A1234A. For this system the code pattern is simple: six characters, the first and the last a capital A and the middle four numerals from 0 to 9. The mask to guard this rule would be “A####A” where the hashtag symbol represents a digit. If, after the bag is treated, the fictional system expects the same four digit code with a B as prefix and suffix, the Macotronic UV application should print a product label with the code “B1234B”. That mask would be “B####B”.

Furthermore, a client may not wish to visualize in the screen a large number of letters and symbols that, albeit important for the digital identification system, convey little-to-no practical information for the human operator. In this case, it would be ideal for the HMI to apply a display filter to hide unwanted characters and exhibit on screen only the ones of interest. In the example above if only digits were desired both primary and treated bags codes, once scanned, would appear as “1234”. The display filter would be a transformation mask defined as “[####[”, where the bracket symbol is an order to ignore any characters present in their position. Table 3 summarizes the symbols used in the masks and their meaning.

Symbol	Signification
?	Any letter from A to Z (both in upper and lower case)
#	Any digit from 0 to 9
§	Any letter OR any digit
[	Suppress the character in that position
Other	All other symbols represent themselves including a space

Table 3 – Masks symbols

Printed at the end of an illumination cycle, the product label contains an encrypted barcode (e.g. “B1234B”) and, just below it, its equivalent human readable code. This code is likewise susceptible to a presentation filter to improve legibility. Ultimately, a single product identification system needs a group of four masks as defined in Table 4. Still, one platelet bag may contain several different ID keys: donation, product and lot codes, each of which may follow different encrypting rules.

Mask	Utility
Input	Entry format for non-treated platelet bags
Display	Screen presentation format
Barcode Output	Barcode format for printed label
Text Output	Human-readable text format for printed label

Table 4 – Mask batch and specific utilities

A treatment center that receives platelet bags from various blood banks must be prepared to accept all alternatives identification codes. This justifies the creation of a mask list.

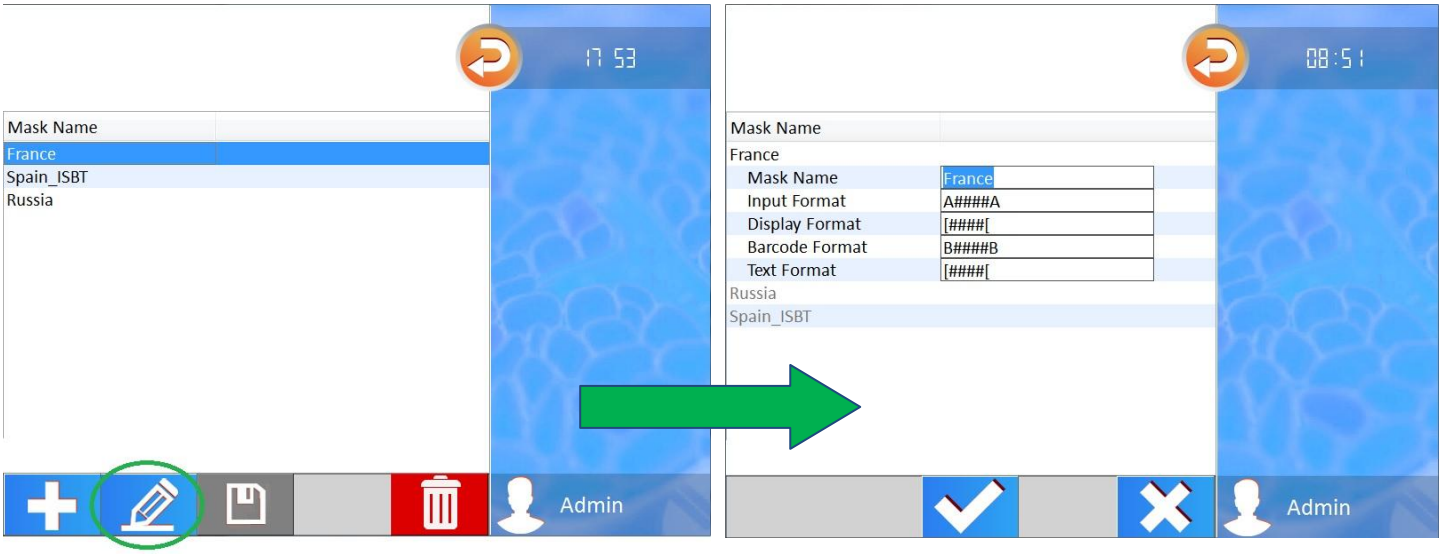

Figure 33 - Mask Management View in editing mode

(Source: own production)

Much like for the management of user information, the application's former version required Macopharma's technicians to register the mask list with an external tool during installation, due to the lack of an embedded dedicated tool.

Profiting from the expandable tree design, the new HMI integrates the Mask Management module the same way it integrates the User Management module (Figure 33). The *MaskManagementView* is another subclass of the *GenericTreeView* thus recycling its behavior. Analyzing Figure 32's diagram, the only substantial change between both module's structure, other than swapping the User Management view and model classes for the Mask Management's, would be the connection with the *MaskModel* instead of the *LoginModel*. Handling all mask-pertaining information, the *MaskModel* manipulates the registered mask list also contained in a XML file.

The reduction in work load for the implementation of the Mask Management module derives from the reuse of the *GenericTreeView* and the *GenericTreeModel* and proves the advantages of a well-thought generic architecture.

4.3.3: Parameter Configuration Module

The correct configuration of the Macotronic UV illumination device is vital to guarantee the quality of the platelet treatment and the user experience. The prototyped application organized the necessary parameters in an Initialization (INI) file that could be opened and edited directly in the Windows system by a Macopharma technician during the installation process. If further modifications were required at a later time, a technician would need to return, no matter how small the alteration.

Once more the issues related to accessing files outside of the HMI application motivated the incorporation of an inlaid parameter management tool. The parameters were structured in a XML file and the *ParametersModel* was implemented to handle its data. However, contrarily to the User and Mask Management modules, this feature does not allow the addition or removal of

parameters, only the modification of their values, observing pre-defined constraints and limiting thresholds. Failure to respect any of these restrictions will result in a warning message, serviced by the *GenericPopupView*.

Since the Parameter Configuration module will only display the parameters' names and values, it was deemed unnecessary the use of the extending tree effect to show numerous fields of information. A simple two column table would suffice for this module's presentation (Figure 34). When in editing mode, the second column will offer either an option box or an editing field, depending on the parameter's characteristic: if it has a list of pre-determined possible values or not. For entering numeric values, the virtual keyboard is at disposal on the right side of the screen (serviced by the *RightBarView*).

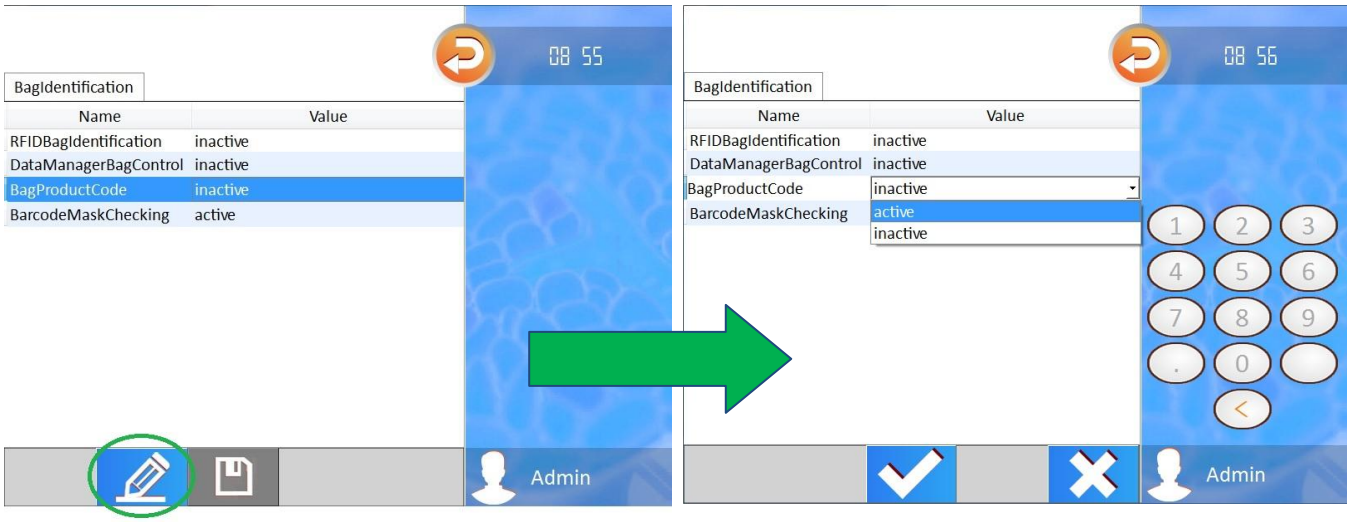

Figure 34 – Parameters View in editing mode (Source: own production)

Even though there were no predictions for other modules to be depicted in the same way, new generic classes (*GenericTableView* and *GenericTableModel*) were implemented for good measure. In future revisions, if there is the need to include a new module with similar characteristics, this course of action would again warrant for a decrease in development time and complexity.

Since the development of this module followed the same principle as the two previously examined, this section does not repeat the relational aspects of its

components. There is, nevertheless, a particularity to this module worth mentioning: not all parameters should be alterable by even an Administrator-level user.

Although some of the parameters are directly related to user comfort and preferences and will not engender problems for the faultless realization of the THERAFLEX UV procedure, others are used for hardware setting and are too sensible and out of an operator's field of expertise to be modifiable by a user.

To ensure that only the right parameters are accessible to users, a "visibility" tag was added for each parameter structure in the XML file. This tag can have either a "private" or a "public" value to define a parameter visible only to a Macopharma technician or to all Administrator-level users, respectively.

A Macopharma technician can log into the system as the MACO user: a special administrator that is always registered and cannot be altered or deleted. The Parameter Configuration module has a connection with the *LoginModel* to inquire about the access rights of the logged user. If it is the MACO user logged, the *ParametersView* will show all existing configuration parameters; otherwise (if it is a normal Administrator) it will show only the publicly-marked parameters. Operator-level users cannot access any of said parameters.

As it is, when a client wishes to change the time it takes for the application to enter the Stand-by mode (power-saving mode), for example, or even wishes to completely disable this option, there is no need to call a specialized Macopharma technician: an Administrator can modify these parameters directly. On the other hand, while modifying these simple configuration elements, there is no risk of this user altering the agitation speed set point for the internal tray's orbital movement (critical to a uniform light exposure of the platelet concentrate bag). Clients have more autonomy while the application gains in safeness and practicality.

4.3.4: Illumination Cycle Module

Launching the illumination cycle is, of course, the most important action a user can perform on the Macotronic UV illumination device. The Illumination Cycle module comprises many essential features and its structure is better detailed in Figure 35. There are three essential views which represent each a major step for performing the THERAFLEX UV procedure.

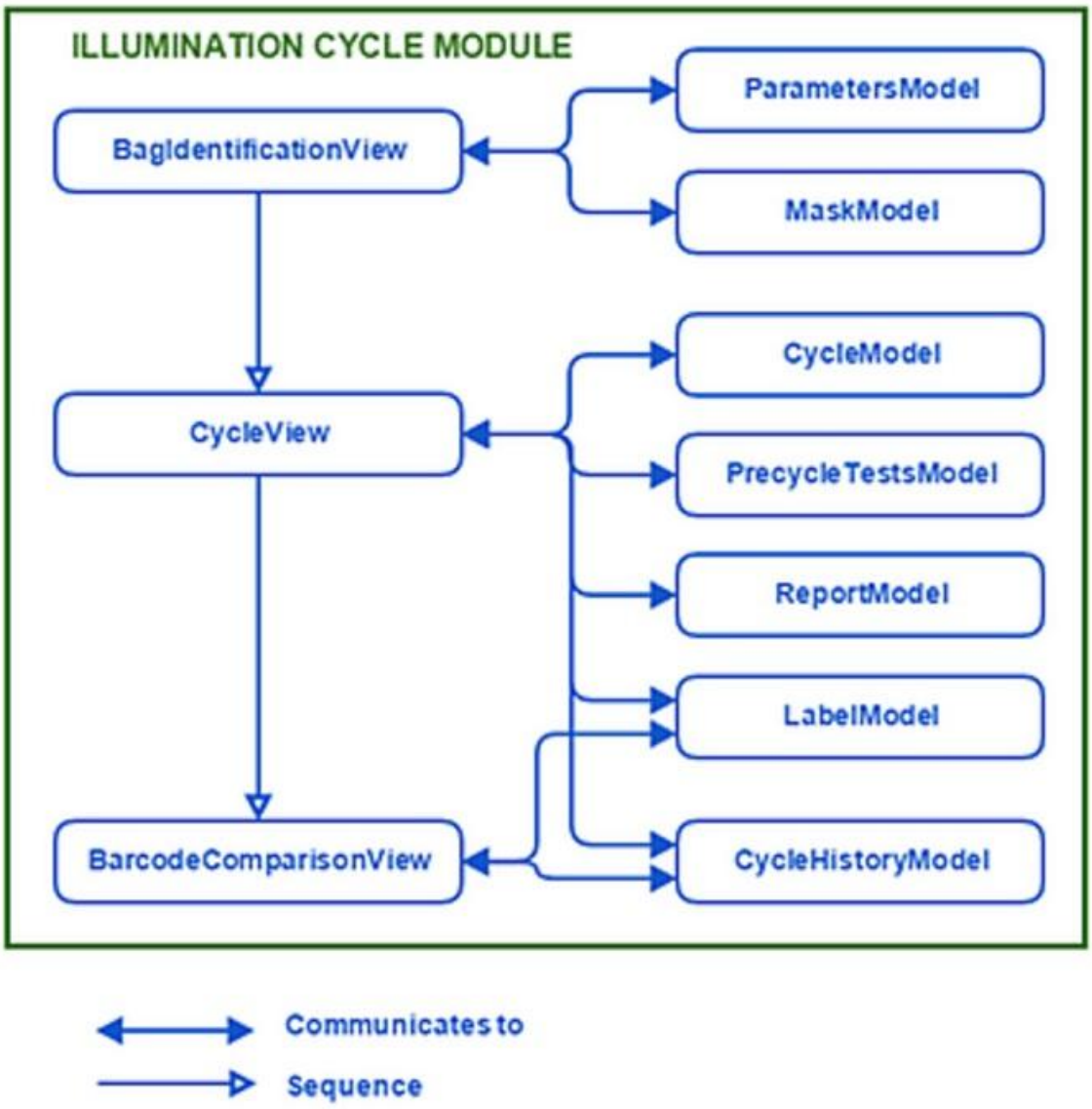

Figure 35 – Illumination Cycle module (Source: own production)

The first step is identifying the platelet bag. Figure 36 partitions the identification process: after placing the platelet concentrate bag on the internal tray (No. 1), the bag's ID codes are scanned (No. 2) and verified against the chosen masks (expected code formats explained in Section 4.3.2:).

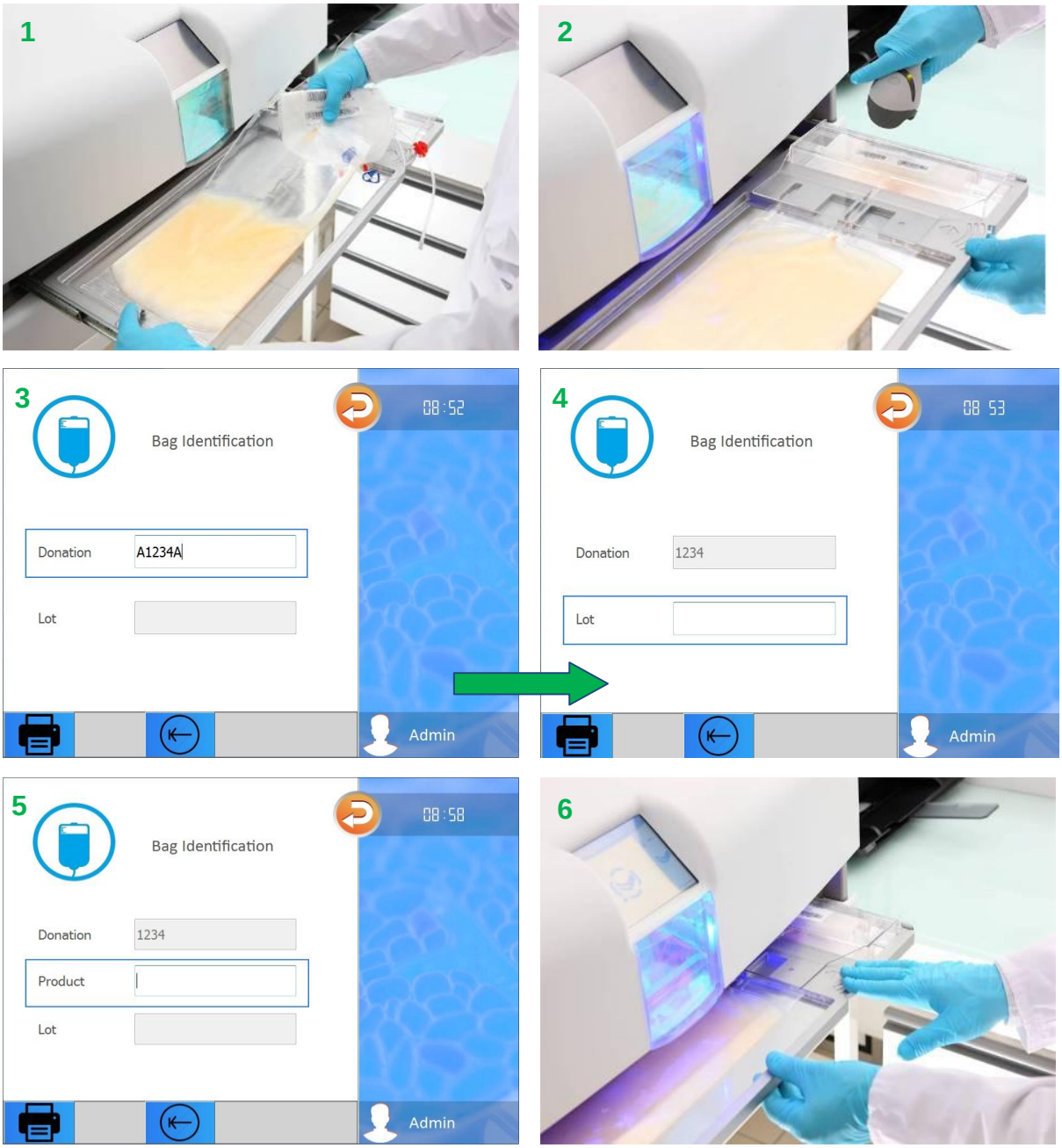

Figure 36 – Bag Identification steps (Source: [10] and own production)

The *BagIdentificationView* displays the scanned codes on the screen (No. 3, 4 and 5). There may be two or three identification keys to suffer validation depending

on the machine's settings: while the donation and lot codes are always present (No. 3 and 4), the product code (No. 5) appears only if so configured (by the parameter highlighted in Figure 34). If and when the validation is successfully completed the user can push the internal tray into the device closing the machine (No. 6) [10].

Once the machine is closed the *CycleView* will be painted on the screen. It pictures a progress bar that will initially represent the advancement of the pre-cycle tests execution and later will be reset to show the cycle's own progression. Figure 37 presents these two situations.

The HMI's pre-cycle tests serve mainly to verify if there is enough space left for the creation of a new cycle history file and if the device's sensors' calibration is not outdated. They recover threshold limits from the *ParametersModel*. In case of a test failure the CycleView will alert the user of the need for space maintenance or machine recalibration. The connections between the *GenericPopupView* and both *CycleView* and *BagIdentificationView*, which accounts for these warning messages, were not included in Figure 35's diagram for simplicity's sake. The same can be said regarding the communication between these two view classes and the *BottomBarView* (the bottom bar buttons can be seen on both views).

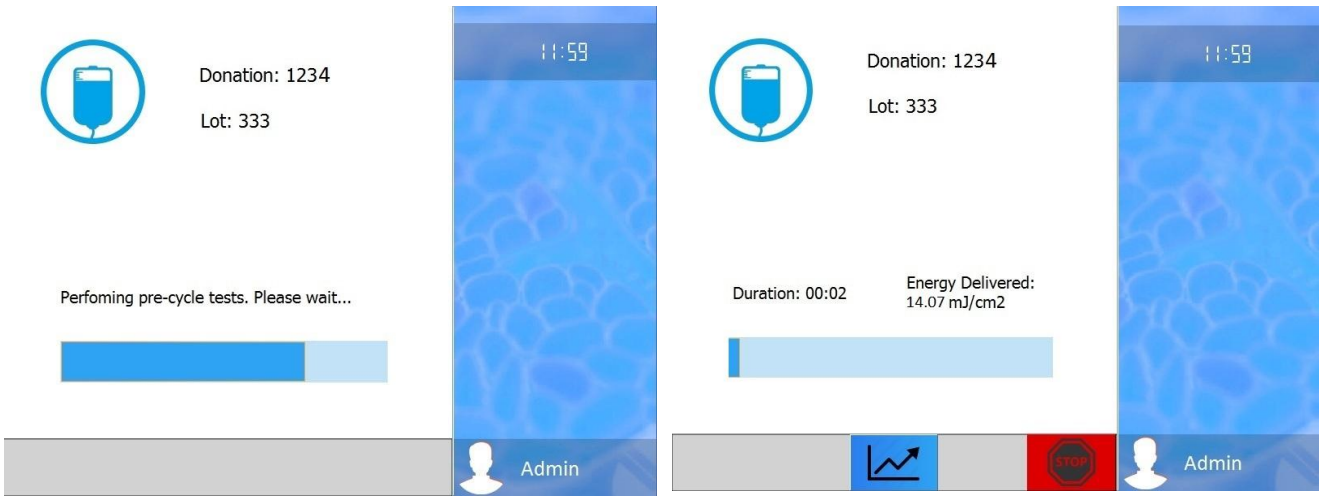

Figure 37 – Cycle View during pre-cycle test (left) and during illumination (right)

(Source: own production)

If the pre-cycle tests are concluded with no errors the *CycleView* employs its model to launch the illumination. The *CycleModel* works with the *MachineModel* and the *CommunicationModel* to send commands via serial communication to the device's microcontroller triggering a hardware response (i.e. turn on tray motor and UVC lamps) thus starting the THERAFLEX UV procedure.

While this operation is running, the *CycleModel* constantly requests sample values from the I/O board: bag temperature, tray's agitation speed, light intensity, delivered energy dose (among others). It stores this data and updates the *CycleView*'s progress bar, duration clock and the accumulated light exposure (Figure 37 on the right). Through the Graph button the view provides the curves of the four aforesaid variables. These features grant users with full process monitoring.

Chapter 5 covers the hardware verifications performed during the cycle by the microcontroller software. Here it is enough to point out that if a hardware mal-function is detected, the cycle will be terminated and an error "popup" message will appear to warn users that the treatment was incomplete. The error code and description will be stored with the rest of the cycle data. The user can also cancel the treatment mid-term by pressing the Stop button, in which case the cycle will be considered as incomplete as well.

Upon receiving an End-of-cycle flag from the microcontroller, regardless of the cycle's status (correctly treated, cancelled or terminated with error), the *CycleModel* signals its view. The *CycleView* will notify the user of the overall treatment result with a green (success: Table 5 - lines 9 to 12), red (suspended: Table 5 - lines 3 to 6) or caution (failure: Table 5 - lines 13 to 18) "popup" view all of which are equally based on the *GenericPopupView* class. Next it will requisition the *CycleHistoryModel* to organize the stored data from the *CycleModel* in a new history file (Table 5 - line 20). Based on this file a report and a product label will then be generated and sent to the connected printers by the *ReportModel* and the *LabelModel* (Table 5 - lines 21 and 22 respectively).

To exit the outcome view and enter the *BarcodeComparisonView* the user must touch any part of the screen so acknowledging the end of the illumination treatment. This final step serves as a product control stage. Reopening the machine,

the user can scan both the original bag codes and the printed code to verify if there were no compatibility problems or impression errors. A Label button on the bottom bar enables label reprinting in case of visibility damage. Once the comparison is validated, the printed adhesive label can be placed on the bag ensuring traceability.

Slot executed in the CycleView when the endOfCycle signal is emitted by the CycleModel	
001	`void CycleView::slt_endOfCycle(bool success) :`
002	`{`
003	`if (a_pCycleModel->getStatus() == CANCELLED)`
004	`{`
005	`a_pGenericPopupModel->setStyle(RED);`
006	`}`
007	`else`
008	`{`
009	`if (success)`
010	`{`
011	`a_pGenericPopupModel->setStyle(GREEN);`
012	`}`
013	`else`
014	`{`
015	`string error = a_pCycleModel->getErrorDescription();`
016	`a_pGenericPopupModel->setStyle(WARNING);`
017	`a_pGenericPopupModel->setDescription(error);`
018	`}`
019	`}`
020	`string filePath = a_pCycleHistoryModel->createHistoryFile();`
021	`a_pReportModel->generateAndPrintReport(filePath);`
022	`a_pLabelModel->generateAndPrintLabel(filePath);`
023	`}`

Table 5 – End-of-Cycle slot

The THERAFLEX UV – Platelets procedure is thus concluded leaving only the need to transfer the treated contents from the illumination bag to the storage bag. The Macotronic UV device is instantly ready to perform another cycle.

4.4: General State Machine Controller

Now that most modules have been explored individually, the general state machine controller can be addressed. This object manages all the application's macro states and the events required to pass from one state to another.

The Controller makes use of Qt's State Machine framework that provides classes for creating and executing state charts: graphical models of how a system reacts to user interaction. Such classes as *QStateMachine*, *QAbstractState*, *QState*, *QFinalState* and *QHistoryState* as well as their embedded functions and signals/slots help to compose and handle the application's dynamics. Figure 38 shows a simplified diagram between these classes and the implemented Controller.

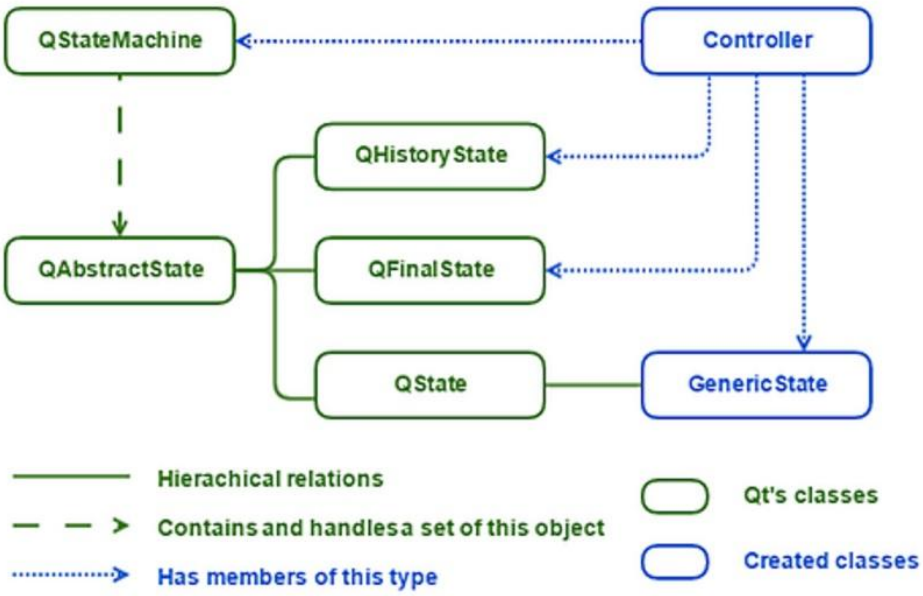

Figure 38 – Controller class (Source: own production)

QAbstractState and consequently all its subclasses, for example, emit a signal every time the state has been entered or exited, which proves most useful in identifying the moment when a specific behavior should be launched. Usually this behavior implies painting a new view on the main window, since a change in state is generally accompanied by a change of the screen.

Functions	Actions Performed
Controller's constructor - called from the main -	Initializes its members: **a_pView = new MainView();** **a_pStateMachine = new QStateMachine(this);** **createStates();**
Controller::start() - called from the main -	Starts the main window and the state machine: **a_pView->show();** **a_pStateMachine->start();**
Controller::createStates()	Creates all states and adds them to the machine: **a_pInit = new GenericState();** **a_pLogin = new GenericState();** **…** **a_pStateMachine->addState(a_pInit);** **a_pStateMachine->addState(a_pLogin);** **…** **a_pStateMachine->setInitialState(a_pInit);** **createTransitions();** **createBehaviors();**
Controller::createTransitions()	Defines all transitions between states: **a_pInit->addTransition(a_pView, SIGNAL(sig_event()), a_pLogin);** **…**
Controller::createBehaviors()	Connects enter-state signals to correct slots: **connect(a_pInit, entered(), SLOT(enterInitView()));** **connect(a_pLogin, entered(), SLOT(enterLoginView()));** **…**

Table 6 – Controller's main functions

QState offers an *addTransition*() function that associates the current state, a signal and its sender to a target state. In other words, it creates a transition between the present state and the target state that will occur only when a given signal emitted by a particular object is received.

The *GenericState* class inherits from *QState* and includes variations of the *addTransition*() function, allowing the event created to be conditional upon the signal's argument. This way, the same signal can trigger transitions to different target states depending on its parameter.

Ultimately when a transition-associated signal is received and the machine changes state the targeted state emits a *entered*() signal which in turn is connected to a slot carrying the desired behavior. Table 6 contains the Controller's main functions and reflects this dynamic.

Slot called when a_pInit state is entered: portrays InitView	
001	**Controller::slt_enterInitView()**
002	**{**
003	**a_pView->setView(NULL, CENTRAL);**
004	**a_pView->setView(NULL, RIGHT);**
005	**a_pView->setView(NULL, BOTTOM);**
006	**a_pView->setView("InitView", FULL);**
007	**}**

Table 7 – Enter-Initialization-View slot

Slot called when a_pLogin state is entered: portrays LoginView	
001	**Controller::slt_enterLoginView()**
002	**{**
003	**a_pView->setView("LoginView", CENTRAL);**
004	**a_pView->setView("RightBarView", RIGHT);**
005	**a_pView->setView("BottomBarView", BOTTOM);**
006	**a_pView->setView(NULL, FULL);**
007	**}**

Table 8 – Enter-Login-View slot

Tables Table 7 and Table 8 hold the implementation of the slots related to the states exemplified in Table 6: *a_pInit* and *a_pLogin*. When the first state is entered the *InitializationView* will be painted on the screen, occupying the whole *MainView* (Table 7 - line 6). As for when the second state is entered the *LoginView* will be set in the central area of the *MainView* (Table 8 - line 3), while the *RightBarView* and the *BottomBarView* will be set respectively on the right side and bottom widgets (Table 8 - lines 4 and 5). So far as no other state-changing events occur, the view class will assume the management of its visual features while the affiliated models will be responsible for their tasks.

Finally, Figure 39 shows a simplification of the chart draw up to organize and sequence the Macotronic UV application's states (in blue) and operations (in green). It defines the user-based and other events that serve as transitions (arrows) between states focusing on the steps to reach the illumination cycle. In order to summarize the graph, other states are simply listed in blue (inside the MenuView) to show that they are featured in the system, rather than detailed in separate boxes.

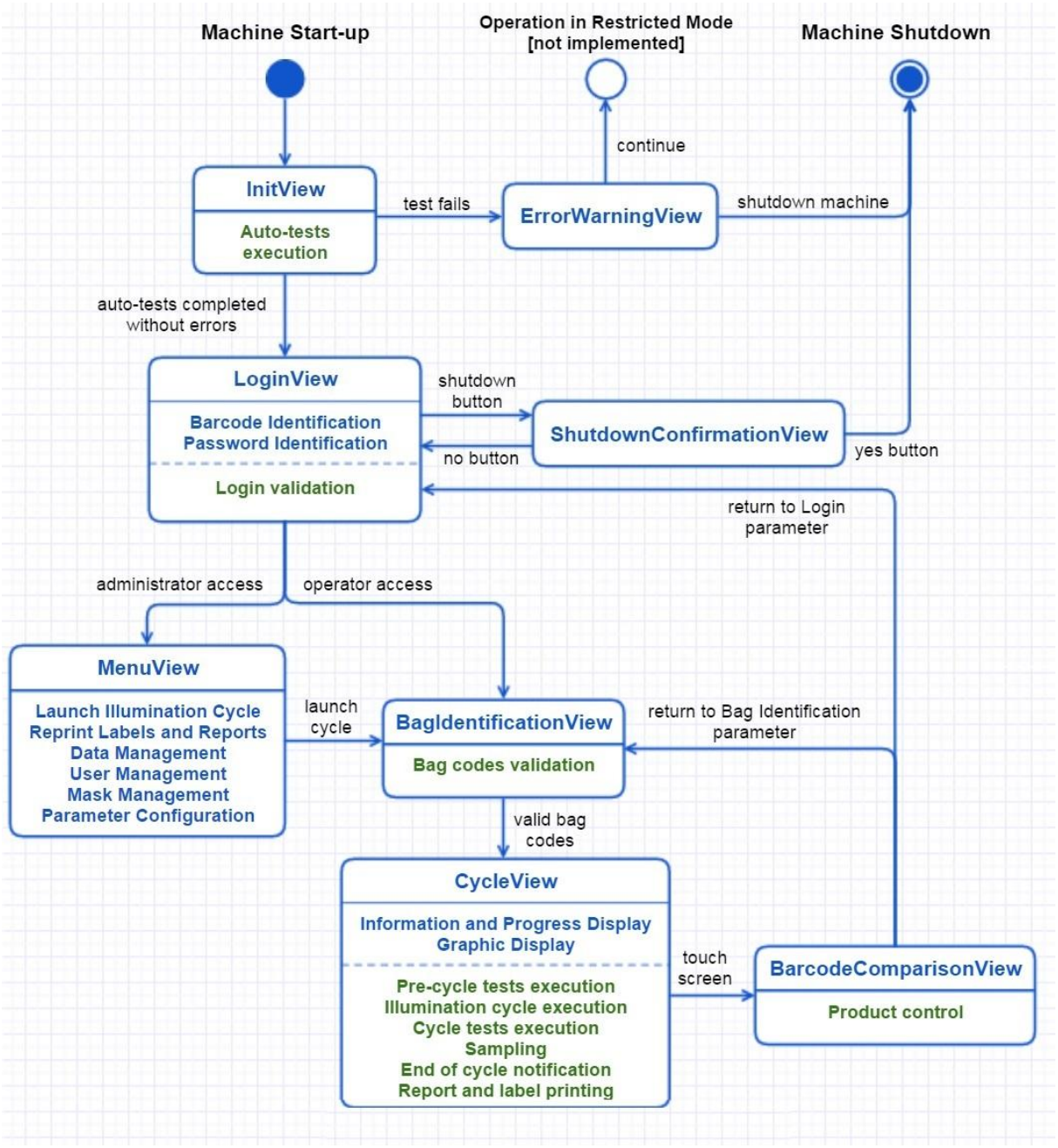

Figure 39 – Macotronic UV's State Machine (Source: own production)

Chapter 5. Microcontroller Software

The Macotronic UV illumination device contains several actuators and sensors: six UVC lamps to illuminate the platelet bag; a relay to switch the lamps on or off; a motor to agitate the internal tray thus supplying the orbital movement needed for uniform light exposure; four photodiodes to detect and quantify the light intensity delivered to the bag; motors to turn on ventilators and cool the machine down; a motor to move up or down the front panel door; end position sensors to verify the door's and tray's placements; and a pyrometer to read ambient and targeted temperature.

Embedded in an analog I/O board (Figure 40) from Digi International's high performance BL4S200 series, the Rabbit® microcontroller provides fast data processing. Its dedicated IDE called Dynamic C accounts for powerful programmability in software development. This chapter focuses on the implemented program that manages the above enumerated components.

Figure 40 – BL4S200 I/O board with the Rabbit® microcontroller

(Source: [7])

To present the basic overall construction of the software Table 9 contains the pseudo-code from its *main*() function. After configuring all digital and analogical inputs and outputs from the I/O board (line 3) the microcontroller continuously reads the photodiodes' data (line 6), verifies and executes any commands sent by the HMI application (lines 7 and 8) and communicates with the pyrometer if necessary (line 9). This fundamental structure has not been modified during the software's revision.

Microcontroller software in C: main function	
001	**void main()**
002	**{**
003	**initilializeIOBoard();**
004	**while(true)**
005	**{**
006	**acquirePhotodiodesMeasurements();**
007	**communicationCPU();**
008	**generalStateMachine();**
009	**communicationPyrometer();**
010	**}**
011	**}**

Table 9 – Microcontroller's main function

Between the former and the new versions of the software the most significant modification was the change in the operating mode regarding the illumination cycle. Originally each action needed to perform a cycle (start tray agitation, turn the UVC lamps on, etc.) was executed individually following direct requests issued by the HMI application: one request equals one action. Currently a single command can set off a chain reaction that will be completely handled by the microcontroller's state machine. This was accomplished by adding new states (notably the *Pre-cycle*, *Cycle* and *Stop Cycle* states pictured in Figure 41), defining their tasks and determining the transitions between them.

Other states were not depicted in Figure 41 simply because no modifications were made concerning them. Although some are obsolete when this program version

is deployed along with the new HMI software, they were maintained to guarantee compatibility with the previous versions and with diagnostically important internal applications.

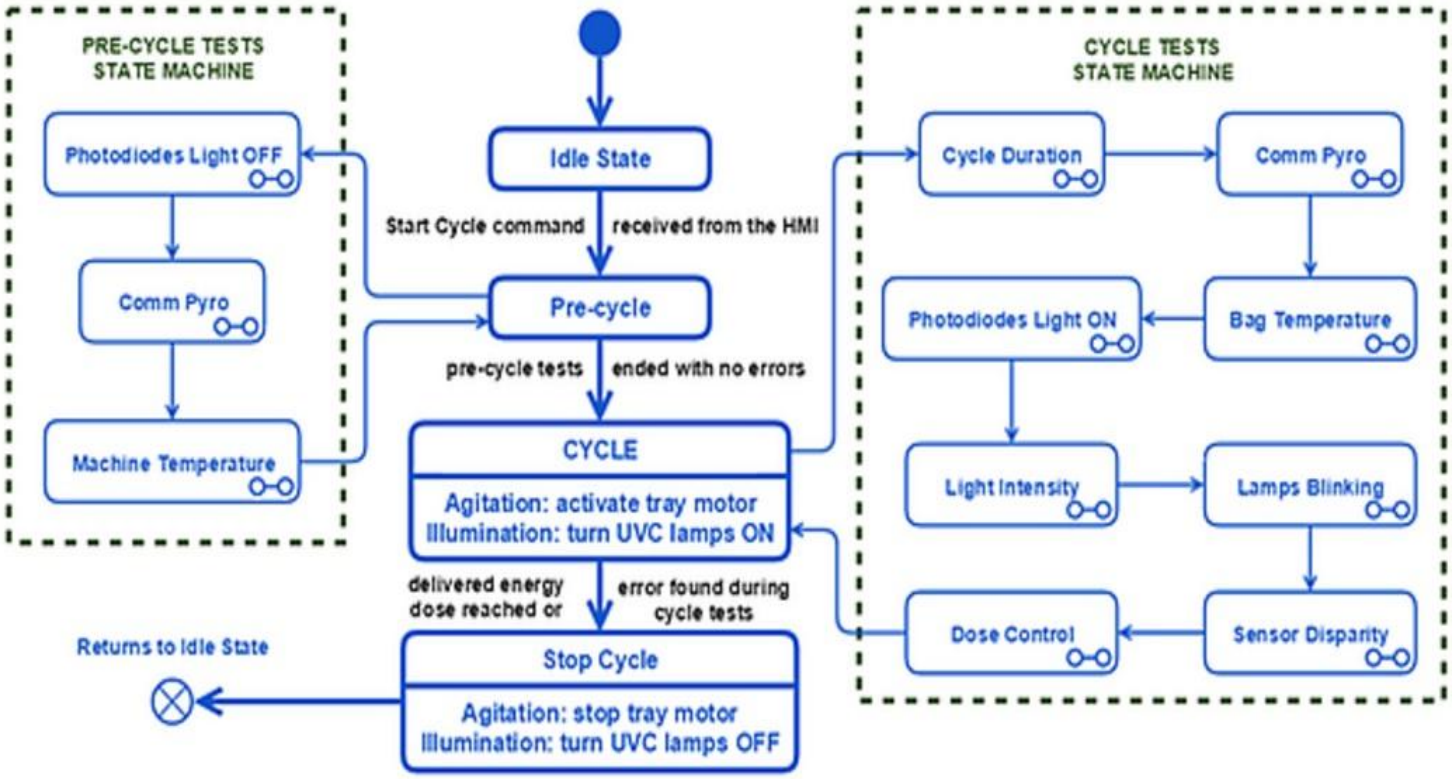

Figure 41 – Microcontroller's General State Machine (Source: own production)

Upon reception of the "Start Cycle" command sent from the CPU the microcontroller will check if the tray is well positioned inside the device and if the front panel door is up and closed. In case these pre-conditions are met, the machine will be set to the *Pre-cycle* state, which is responsible for executing hardware tests before the start of the cycle to ensure there are no problems of communication with the pyrometer, that the machine's temperature is at an acceptable value and that all UVC lamps are turned off (thus not already emitting light which would not be considered in the final delivered energy dose, compromising its precision).

A failure in only one of these verifications is enough to change the machine's state back to *Idle*, preventing the illumination process from happening. If, however,

all tests are successfully concluded the next state is attained. The *Cycle* state will activate the tray motor, starting the orbital movement. It will verify agitation stability issues to increase device and user safety. Once speed consistency is reached, the UVC lamps will be turned on, officially decreeing the start of the THERAFLEX UV treatment.

Other hardware verifications will be regularly performed for the whole duration of the cycle. These tests serve to detect disturbances, disruptions, sudden collapses or incapability among other issues that may befall the device's components. They are briefly explored in the next sections. In the event of an error the machine will pass to the *Stop Cycle* state, terminating the treatment before its conclusion.

As long as no problems are uncovered and at each passage through the *Cycle* state, an examination of the cumulated delivered energy dose will also be executed in order to identify the end of the procedure. When an ideal energy dose is achieved, the treatment is considered complete and the *Stop Cycle* state takes over, turning the UVC lamps off, deactivating the agitation motor, opening the front panel door and ejecting the tray. Lastly, it will return the state machine to its idle status to wait for new cycles.

After triggering the illumination procedure, the only other interaction with the HMI will be answering its request for sample values during the cycle. This section does not linger in the particulars of the HMI-microcontroller relationship, since they are covered later in this chapter.

5.1: Cycle Tests

Previously performed by the HMI application, the tests were included in the microcontroller software to optimize the process and decrease the constant message flow between the CPU and the microcontroller. These verifications compose a sub state machine where each test is represented by a state, as shown in the right side of Figure 41.

Usually a test consists of comparing sensor measurements to reference values entered as parameters. The blueprint of the tests' basic operation can be observed in Table 10. Handled by the HMI application as discussed in Section 4.3.3:, the mentioned parameters are sent from the CPU to the microcontroller at machine start-up to configure set points and thresholds.

The sub state machine for the execution of the cycle tests	
001	`void cycleTests()`
002	`{`
003	`switch(test)`
004	`{`
005	`case CYCLE_DURATION:`
006	`{`
007	`if ((currentTime - cycleStartTime) > maxCycleDuration)`
008	`{`
009	`cycleError = CYCLE_DURATION_ERROR;`
010	`}`
011	`test = COMM_PYRO;`
012	`break;`
013	`}`
014	`…`
015	`}`
016	`}`

Table 10 – Basic structure of a cycle test

If the measurements are found to be out of the expected range the test fails and an error code specific to the problem is registered in a variable. The same variable is monitored by the general state machine to terminate the illumination cycle in case of a fault. The error code is sent to the HMI application so it can show a description of the problem to the user (see Table 5 lines 15 to 17 in Section 4.3.4:). This facilitates diagnostics and decreases the maintenance team's work load.

5.1.1: Cycle Duration

As the simplest test on the list, this verification compares the time elapsed since the beginning of the cycle, when the UVC lamps are turned on, to the maximal duration value. It is used as an example in Table 10. A cycle too long reflects a problem reaching the illumination dose: maybe not all lamps were properly switched on or the lamps are not emitting enough light.

5.1.2: Pyrometer Communication

The device's existing pyrometer is a non-contact infrared thermometer of the Optris® CS series. It calculates the surface temperature of an object based on its emitted infrared energy. It is a sensitive optical system with a housing made of stainless steel that contains the complete sensor electronics. To supply the measured temperature values, it communicates with the microcontroller by means of a bidirectional RS232 connection [8].

Using its own communication protocol, defined in its Operators Manual [8], a command is sent through the serial line to request the targeted object's temperature: in this case the object is the platelet bag. If there is a problem with the communication, for example the command is either not received or not treated by the pyrometer, this test fails. Otherwise the sub state machine switches to the next test which will use the value provided by the sensor.

5.1.3: Bag Temperature

The platelet concentrate bag is stored in low temperatures for preservation. However, for the THERAFLEX UV procedure to achieve its goal, the bag must be

warmed before undergoing illumination. The energy emitted during the process undoubtedly heats the bag (up to 2 ºC more than its initial temperature), but overheating can irreversibly damage platelet structure. With this in mind it is evident that the platelet bag's temperature must stay within an interval delimited by a minimal and maximal threshold (normally 18 ºC and 32 ºC respectively).

The test converts the value received from the pyrometer to a temperature in degrees Celsius and examines if the result is inside that interval. Table 11 shows the conversion rule as it is given in the Optris® CS Operators Manual [8]. While this test compares the targeted object calculated temperature, the Machine Temperature pre-cycle test, present in Figure 41, uses the value read from the pyrometer's head and interprets it as the Macotronic UV device's ambient temperature.

Pyrometer's answer	Result in ºC
Target Temperature (bag): byte1byte2	**(byte1 x 256 + byte2 - 1000) / 10**
Head Temperature (machine): byte1byte2	**(byte1 x 256 + byte2 - 1000) / 10**

Table 11 – Conversion equations for bag and machine temperature

5.1.4: Photodiodes Light On

This test verifies that all UVC lamps are functioning correctly by comparing the measurements of each photodiode to a value that defines the minimal amount of light emanation needed to consider a lamp as turned on. Due to the lamps' and photodiodes' dispositions, if a particular set of values is found to be under the determined limit, the photodiodes implicated indicate which lamp is likely turned off.

The Dynamic C library provides an inlaid function to read analogic inputs and transform them into digital information. This function is called to acquire the photodiodes' data converting its 0 – 10 V entries to a 0 – 2047 nondimensional range [7]. Before performing the verification the test has to process the data into mW/cm²: the unit in which the reference values sent by the HMI application are given. Since 0 – 10 V is equivalent to 0 – 100 mW/cm² [9] the transformation rule is:

measurement in mW/cm² = photodiodeValue * 100 / 2047

This equation is used every time the values read by photodiodes are needed in a calculation to perform a test (as is the case in all verifications described in the following sections).

5.1.5: Light Intensity

To guarantee a stable illumination, this test verifies if the emanated light intensity is comprised in a set scope. It calculates the average of all photodiode-acquired values and compares it to the interval's boundaries. If the average falls outside the limits it is recalculated with newly measured values until either the comparison succeeds or a timeout occurs. The test fails if and when the timeout occurs; meaning that for a certain period of time the average light intensity was constantly out of scope.

5.1.6: Blinking Lamps

As its name suggests, this test detects flickering lamps. To do so, it calculates a ratio between the current and the preceding measurements for each photodiode and compares it to a referenced tolerance. Significant discrepancies between present and previous acquisitions result in a high ratio value and indicate that a lamp is shining intermittently or unsteadily. Every time the ratio is greater than the tolerance an error counter is incremented and the comparison is repeated with freshly procured values. A successful comparison implies a reset of the error counter. To prevent raising a false alarm owing to acquisitioning problems, the test will only fail if the error counter reaches a parameterized number, denoting

consecutive variance. Again, the photodiodes affected pinpoint which lamp is mal-functioning.

5.1.7: Sensor Disparity

In order to identify faulty photodiodes this test compares the measured values of each photodiode individually with the average of all values. Disparities more substantial than the pre-defined tolerance range uncover an issue with the photodiode's sensing capacities.

5.1.8: Dose Control

Contrarily to previously visited tests, this control does not detect hardware related problems, but rather serves to ascertain the end of the illumination cycle. It calculates the accumulated light intensity by adding at every pass the former to the current delivered energy dose. The dose is computed with the multiplication of the photodiodes' average and the time elapsed since the last calculation took place.

The Stop Cycle state is attained once the cumulated dose reaches or surpasses the set point. This state will turn the UVC lamps off and stop the tray's agitation as described in the first section of this chapter. Seen as the lamps take a little time to completely stop emanating light, another test is performed to inspect the photodiodes' acquisitions (in case they are still pick up any input signals) and eventually consider the residual light, verifying that in total the delivered dose does not exceed a maximal threshold, which could endanger platelet structure and functionality and invalidate the whole procedure.

5.2: Communication Protocol

The interactions between the HMI application and the microcontroller are accomplished with a RS232 serial connection through which messages containing characters in the American Standard Code for Information Interchange (ASCII) are exchanged. Table 12 shows the main configuration characteristic for the communication interface [11].

Communication Protocol Configuration	
Baud rate	**115200 bauds**
Data bits (per character)	**8 bits**
Parity	**None (N)**
Stop bits	**1 bit**
Flow control	**Off**

Table 12 – Serial connection settings

Being a bridge between the two software components of the Macotronic UV device it stands to reason that the communication protocol is build based on tasks performed by both of its ends. As it was conceived for the prototype version, the CPU board serves as a master while the I/O board is considered the slave. In practice this means that the microcontroller can only respond to requests from the HMI and not pro-actively send messages to the application.

On the HMI's side, the *CommunicationModel* employs a *QSerialPort* object (provided by Qt) to find the correct serial port and set it with Table 12's parameters. It then stablishes a connection by opening this port. It is also responsible for constructing and sending demand frames to the I/O board. At the other extremity the microcontroller software will read these demands and inspect the data's structure before executing the commands and returning the awaited answers.

The transited frames' composition follows a defined and unchangeable arrangement (Table 13): a start byte to determine the beginning of the frame; a command byte consisting of the request's identification key; optional data bytes to

eventually supply the microcontroller with important information; a checksum byte that varies according to the precedent bytes' values and is used to detect accidental changes in data; a stop byte to recognize the end of the transit container.

Name	Representation	Hex - Dec	Example: Ping cmd
Start byte	**:**	**3A - 58**	**:**
Command byte	**CC**	**(variable)**	**30**
Data bytes (optional)	**DD**	**(variable)**	**01020304**
Checksum byte (CRC)	**XX**	**(variable)**	**12**
Stop byte (CRLF)	**\r\n**	**DA - 218**	**\r\n**

Table 13 – Command basic structure

A Cycle Redundancy Check (CRC) code was used as the checksum byte because it is simple to implement in binary hardware, easy to analyze mathematically and particularly efficient at uncovering common errors caused by noise in transmission channels. The *CommunicationModel* calculates a fixed-length CRC value based on the command byte and eventually the data bytes and appends it to the message as a redundancy: it expands the message without really adding new information to it).

Upon reception of a frame, the microcontroller software will initially ascertain if the first and last byte correspond respectively to a ":" and a "Carriage Return - Line Feed" (CRLF) character. If the start and stop bytes are validated, a corroboration of the CRC value will take place: the received data is used to calculate a fresh CRC which is then compared to the received checksum byte. In case of a match the message's integrity is ensured and the command can be executed, otherwise an error code is sent back to the CPU. Other errors may occur during frame treatment producing different responses: Table 14 presents the possible resulting codes.

Name	ID (Hex)	Function
ACK_SUCCESS	**0x0F**	No error found during data processing
CHECKSUM_ERROR	**0xF2**	No match of CRC values
CMD_UNKNOWN	**0xF3**	Command identifier unknown
FRAME_SIZE	**0xF4**	Frame size differs from expected

Table 14 – Possible acknowledgement messages

Both the HMI's and the microcontroller's programs have an equivalent list of identification codes for the exchangeable requests. Considering the alterations in the operating mode explored in earlier in this chapter, new ID keys were included in these lists. The Ping command, exemplified in Table 13, was added so that the application could continuously verify the status of the connection (no reply received indicates communication failure). The implications of the other requests listed in Table 15 have been described in the beginning of the present chapter.

Name	ID	Function
PING	**30**	Questions if microcontroller is listening
CONFIG	**31**	Set reference values for cycle (tests and set points)
START_CYCLE	**32**	Trigger cycle's chain reaction (pre-tests, agitation, illumination)
CYCLE_SAMPLE	**33**	Request sample values from illumination cycle
STOP_CYCLE	**34**	Terminates cycle forcibly (user's cancel cycle action)

Table 15 – Added commands

Although the new HMI application still profits from some other commands, notably the ones to open/close the front panel door and eject/insert the tray, most of the original requests became obsolete for this revision's operation. However, to maintain compatibility with former versions and to allow a connection between the microcontroller and a diagnostics application – called UVC Test and developed in parallel by another member of the Automation's team – all the existing demands and their consequent actions were kept intact in the microcontroller's software.

Chapter 6. Conclusions

The Macotronic UV project was carried out to reengineer the embedded system of the prototype version, improving user comfort, enhancing code maintainability, optimizing process control and increasing the illumination precision, thus achieving overall higher quality levels for the THERAFLEX UV-Platelets pathogen inactivation procedure.

Taking into account client feedback, a completely new Human-Machine Interface was developed. This application, focused in sophisticated looks and user-friendly, intuitive features, aimed to simplify tasks and, with its inlaid configuration tools, avoid problems arisen from installation errors. The software was designed with a modular architecture in order to facilitate future revisions and reduce maintenance-related efforts. The graphic arrangement, however, has yet to be validated by Macopharma's Marketing department.

The modifications made in the microcontroller's operating mode allowed for a single user command to trigger a sequential flow of actions needed to perform an illumination cycle.

With fewer messages exchanged between the CPU and the I/O boards and by locally calculating the cumulated delivered energy dose, the microcontroller software renders a more accurate illumination with regards to the total light intensity required to inactivate leukocytes and pathogens present in the platelet concentrate bag without damaging platelet structure and function. The internal hardware verifications currently executed by the microcontroller speeds the detection and identification of components' failures and no longer require direct interaction with the main application.

To accommodate the aforementioned changes in operation, the communication protocol between the user interface and the microcontroller software was adapted and new requests were included. Nevertheless this interface remained otherwise intact to prevent compatibility issues with previous versions of the system.

During the development of the Macotronic UV project, all software implementations followed internal standardizations and good programming practices provided by the Automation department’s quality guides.

Although at each development stage the corresponding tests were performed to assess if the implemented features respected the associated specifications, a global validation will take place before the official release of the Macotronic UV embedded system’s new version.

Due to the short time of the internship, there are still some features to be implemented before the project’s completion, notably the hardware auto-tests that must be executed during the machine's start-up to ensure that all the device’s components are active and ready to use. There are also finishing touches and program adjustments to be made to achieve an increase in code robustness.

Finally this project represents the first step in releasing an innovative machine that can potentially help health providers around the world to deliver a purer and safer treatment for those in need of platelet transfusion.

Bibliography

[1] Sustainability Report (2014). Macopharma's Internal Press Release.

[2] Alberts, B., Johnson, A., Lewis, J. (2002). *Molecular Biology of the Cell* (4th edition). New York: Garland Science.

[3] Chu, RW. (1999). Leukocytes in Blood Transfusion: Adverse Effects and Their Prevention. *Hong Kong Medical Journal*, 5, 280-284.

[4] Seghatchian, J., Tolksdorf, F. (2012). Characteristics of the THERAFLEX UV-Platelets Pathogen Inactivation System. *Transfusion and Apheresis Science,* 46, 221-229.

[5] Model/View Programming (no date). *Qt Reference Documentation.* Retrieved from http://doc.qt.io/qt-5/model-view-programming.html (last access January 8th 2016).

[6] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley Professional.

[7] *BL4S200 User's Manual* (2008 - 2010). Digi International Inc.

[8] *Optris CS Operators Manual* (no date). Optris.

[9] *PLC UV/VIS Sensor Datasheet* (no date). Dr. Gröbel UV-Elektronik GmbH.

[10] *Macotronic UV User Manual* (2012). Macopharma.

[11] *Macotronic UV Software Design Specification* (2015). Macopharma.

[12] Redmine Features (no date). *Redmine Documentation.* Retrieved from http://www.redmine.org/ (last access December 5th 2015).

[13] About Tortoise SVN (no date). *Tortoise SVN Documentation*. Retrieved from https://tortoisesvn.net/ (last access December 5th 2015).

[14] Qt Creator Manual (no date). *Qt Reference Documentation*. Retrieved from http://doc.qt.io/qtcreator/ (last access January 7th 2016).

[15] Qt Designer Manual (no date). *Qt Reference Documentation*. Retrieved from http://doc.qt.io/qt-5/qtdesigner-manual.html (last access January 5th 2016).

[16] Qt Linguist Manual (no date). *Qt Reference Documentation*. Retrieved from http://doc.qt.io/qt-5/qtlinguist-index.html (last access January 18th 2016).

[17] Qt qMake Manual (no date). *Qt Reference Documentation*. Retrieved from http://doc.qt.io/qt-5/qmake-manual.html (last access January 18th 2016).