MR Platform 2.0

By

Mrudul Y Doshit

14MCEC07

Guided By

(Internal)

Prof. Jaladhi Vyas

Assistant Professor, CSE Department

(External)

Abinash Ojha

Senior Technical Specialist, MR Imaging, Philips Healthcare India

Sajith Satheesan

Senior Project Manager, MR Imaging, Philips Healthcare India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

MR Platform 2.0

Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer science & Engineering

By

Mrudul Y Doshit

14MCEC07

Guided By

Prof. Jaladhi Vyas

Department Of Computer Science & Engineering

Abinash Ojha

Senior Technical Specialist, MR Imaging, Philips Healthcare India

Sajith Satheesan

Senior Project Manager, MR Imaging, Philips Healthcare India

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

AHMEDABAD-382481

Certificate

This is to certify that the Project entitled "MR Platform 2.0" submitted by Mrudul Y Doshit (14MCEC07), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science & Engineering of Nirma University, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this Project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Jaladhi Vyas
Guide & Assistant Professor,
Department of Computer Science & Engg,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Priyanka Sharma
Professor & Coordinator (M.Tech - CSE),
Department of Computer Science & Engg,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Sanjay Garg
Professor and Head,
Department of Computer Science & Engg,
Institute of Technology,
Nirma University, Ahmedabad

Dr. P.N Tekwani
Director,
Institute of Technology,
Nirma University, Ahmedabad

Certificate

This to certify that Mrudul Y Doshit (14MCEC07), a student of M.Tech in Computer Science & Engineering, Institute of Technology, Nirma University, Ahmedabad has been working in this organization since the 2nd of July, 2015 and has carried out his project work titled "MR Platform 2.0". He is working as an intern under the supervision of Abinash Ojha (Mentor), and Sajith Satheesan (Manager). He has successfully completed the assigned work and is allowed to submit his project report. The results embodied in this project, to the best of our knowledge, havent been submitted to any other university or institution for award of any degree or diploma.

Abinash Ojha
Senior Technical Specialist,
Magnetic Resonance Imaging ,
Philips Healthcare
Bangalore

Sajith Satheesan
Senior Project Manager,
Magnetic Resonance Imaging,
Philips Healthcare
Bangalore

Statement of Originality

I, **Doshit Mrudul Yogeshbhai**, Roll. No. **14MCEC07**, give undertaking that the Major Project entitled "**MR Platform 2.0**" submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made. It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Signature of Student

Date:

Place:

Endorsed by

Prof. Jaladhi Vyas

(Signature of Guide)

Acknowledgements

First and foremost, sincere thanks to my mentor, Abinash Ojha, (Technical Specialist, MR Imaging). I enjoyed his valuable guidance and inputs and owe him lots of gratitude for having a profound impact on this report. I would like to thank my teammates Azhar Khan and Rabindranath Mishra for their vast knowledge, without which, this project work would never have been completed. It gives me immense pleasure in expressing thanks and profound gratitude to Prof. Jaladhi Vyas for giving me an opportunity to work under his guidance. I would also like to thank my manager Sajith Satheesan for his immense support and encouragement. It gives me an immense pleasure to thank Dr. Sanjay Garg, Honble Head of Computer Science and Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his kind support and providing basic infrastructure and healthy research environment. A special thank you is expressed wholeheartedly to Dr P.N Tekwani, Honble Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation he has extended throughout course of this work. I also owe my friends and colleagues at MR Department, Philips Innovation Campus, special thanks for helping me on this path and for making this internship more enjoyable. I would like to thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work. At last I would like to thank Nirma University and Philips Healthcare for providing resources and quality environment for research and development.

- Mrudul Y Doshit

14MCEC07

Abstract

Magnetic Resonance Imaging (MRI) Systems are complex medical diagnostic machines having components such as the magnet, gradient coils, shim coils, etc. These components are managed by a computer, which also hosts the applications and services required to use the system. These applications and services should reliable, accurate and should work in real-time, as a small mistake may have life-threatening implications. This project report describes the development of a distributed messaging framework to be used by MR applications to communicate on a distributed platform, as well as a set of tools used to improve the efficiency, reliability, and predictability of the system.

A typical MRI system contains a wide array of processes and services, which require inter-communication. The distributed messaging framework is a middleware that enables message-based communication among different platforms (Linux-based, Windows, VSWorks) and technologies(C++/C# Applications, Web Applications). It provides a an efficient, reliable and error-resilient solution and provides REQUEST-RESPONSE, ROUTER-DEALER (Asynchronous) and PUBLISH-SUBSCRIBE communication modes.

Philips has a huge portfolio of MRI systems, which include 40+ releases supporting 800+ configurations. Philips currently provides support for around 34 releases, has around 12 products in production and around 5 products are under development. Maintaining such a huge number of releases and configurations is an uphill task. This project report also describes a set of tools created to simplify and automate certain aspects of installation and updation, and configuration, as well as to perform system analysis and generate reports indicating vital system parameters and gauge the health of system components and resources, with the basic aim to ameliorate the current arduous and time consuming systems in place.

Acronyms

- **MQ : Messaging Queue**
- **WebMQ : Web-Based Messaging Queue**
- **MRI : Magnetic Resonance Imaging**
- **SP : Service Pack**
- **RSI : Remote Software Installation**
- **CTA : Configuration Test Automation**
- **FSE : Field Service Engineer**

Contents

List of Figures

Chapter 1

Introduction

1.1 Company Profile

Koninklijke Philips N.V. (Royal Philips) is a Dutch company founded in 1891, primary listed on the Euronext Amsterdam stock exchange. Currently headquartered in Amsterdam, the capital city of the Kingdom of the Netherlands, Philips consists of three divisions/Holding Companies:

- Philips Healthcare
- Philips Lighting
- Philips Consumer Lifestyle

Philips Healthcare is a company committed to providing meaningful innovations that aim to improve the quality of care, enhance patients lives and enable the delivery of better outcomes at a lower cost. Philips Healthcare is a Philips Healthcare is a global leader in patient monitoring, cardiac care, home healthcare solutions, and related customer services, headquartered in Best, Eindhoven, Netherlands. Philips Healthcare is a global leader in medical imaging equipments that include MR (magnetic resonance), and CT (computed tomography), X-ray and ultrasound scanners. A comprehensive list of Philips Healthcare products is given below:

- Magnetic Resonance
- Ultrasound
- Radiography
- Mammography
- Fluoroscopy
- Diagnostic ECG
- Hospital Respiratory Care
- Emergency Care & Resuscitation
- Interventional X-Ray
- Clinical Informatics
- Computed Tomography
- Advanced Molecular Imaging
- Home Healthcare

Philips has a presence in over 100 countries, spanning 6 continents; and has research facilities in India, The Netherlands, USA, UK, Germany, France and China. The R&D Facility in India is located in Bangalore, and is known as the "Philips Innovation Campus". Established in 2000, PIC employs over 2000 medical professionals, engineers and researchers; developing intelligent connected products, services and apps, that offer richer experiences for consumers. The Magnetic Resonance Imaging (MRI) is the largest department at PIC, with a team of about 150 people, which owns and develops most of the MR software at Philips.

1.1.1 Magnetic Resonance Imaging (MRI)

Philips MRI product line includes more than 15 machines having a power range of 1.5 Tesla, 3 Tesla and 7 Tesla, supporting applications such as hospital diagnostics, therapy, clinical research, etc. It also provides mobile MR units.
Magnetic Resonance Imaging (MRI) is a technique for generating images of body parts/organs by placing the body in a strong magnetic field and measuring the response of the atomic nuclei of body tissues to high-frequency waves (radio) applied to the specific area of the body. An MRI scan provides as much information about the body part/organ as a CT Scan, an Ultrasound and an X-Ray combined. An MR scan also shows some aspects than cannot be seen by using other imaging methods. Some of the problems detected by an MR scan include injuries, identifying flow of blood and diseases/blockades to blood vessels, tumors, etc. An MRI can provide a comprehensive image about the brain, which cannot be matched my any of the other scanning methods available. Some of its advantages include:

- The ability to perform multiple imaging concurrently.
- The ability to scanning any part of the imaging plane without having to physically move the patient.
- A wide range of available soft tissue contrasts, thus leading to a detailed anomaly detection. In case of brain, this property can be used to sensitively identify specific abnormalities within the brain.
- MRI allows the evaluation of structures that may be obscured by artifacts from bone in other imaging methods like the CT Scan.
- MRI contrast agents have a considerably smaller risk of causing potentially lethal allergic reaction.
- MRI does not use ionizing radiation, and is thus preferred over a CT scan in children.

In addition to diagnostics, MR is also used for therapy. MR Therapy can be used to kick-start reparative processes in the cells and tissues in a specific region. It can also be used to treat diseases like cancer, and may be used in place of the painful process of chemotherapy in the future. The magnetic field strength used in MR Therapy is approximately 10 times weaker than that used in diagnostic MRI.

1.2 Team Profile

The MRI SW-Infra Team is a part of MR Imaging at Philips Healthcare. The team acts as an interface between the hardware layer (Platform Team) and the application layer (Console, Recon, PatAdmin, etc). The team's core tasks include handling the kernel-level tasks, managing various processes running on the system and provide interfaces that can be used by other Applications/Processes. The major components of infra include:

- OS And Interfaces : Philips uses a customized striped-down version of Windows Operating System as the core, and MR applications run on the top of it. Infra is responsible for managing the OS and providing interfaces to the upper layer applications/processes so as to ensure platform independence. Interfaces also provide a layer of abstraction for the MR system processes.

- MR Process Management : Process management (including managing when to invoke, their lifetime, resource management, etc) is the responsibility of infra team, a fraction of which which is managed by the OS itself.

- Logging framework : The Logging Framework provides a centralized logging mechanism for all the applications/processes running on the system.

- Messaging Framework : The Messaging Framework provides a centralized environment for MR processes to communicate with each other.

- Configuration Framework : The Configuration Framework manages the components and the configurations of an MR machine, based on the type of the system, the hardware configuration and the licensing details. It also manages the authentication and licensing aspects of the system.

- Service Pack Framework : The Service Pack Framework is used to push patches, updates and new functionality to the older and newer releases in the form of Service Packs.

1.3 Problem Statement

One part of the problem statement is the need of a modern high performance messaging framework. The current system in place is old, archaic and doesn't provide advanced communication modes. Moreover, it uses windows-native sockets for communication, which makes it unsuitable to be used on multiple platforms. It also doesn't support web applications. Hence the need for a new messaging framework that s modern, uses the message-queue concept for better performance, and supports multiple platforms and languages. The other part consists of building automation tools used to improve the efficiency of Philips MRI systems and its related processes. Philips has been building MRI systems since 1980. Over the years, it has had a number of releases (40+ releases), supporting a large variety of configurations (800+ configurations). Of these, Philips still provides support for around 34 releases, has around 12 products in its production portfolio and around 5 products are in the development phase. Maintaining such a huge number of variants is an incredibly complex task. Owing to such a huge pool of releases and configurations, even a small change requires a complex set of tasks to be performed, and involves a lot of repetition. Thus, the need for a set of tools that provide the improvements and automation and which can be used for any release, for any configuration on any MR machine that Philips provides support for.

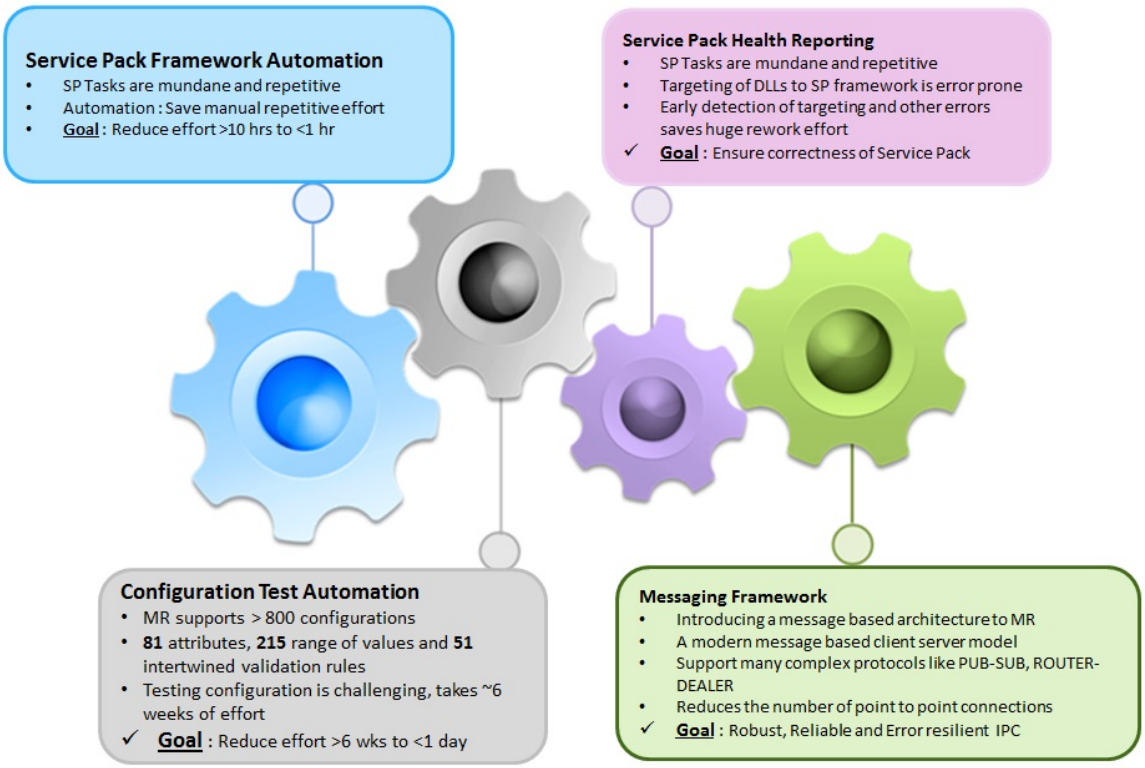


Figure 1.1: An Overview Of The Work Planned During The Internship

1.4 Project Purpose And Scope

One part of the project objective is to provide a distributed messaging framework to be used by MR applications to communicate on a distributed platform. the framework should have low latency, high performance, error-resilience and should have a generic interface that supports multiple technologies (C++/C# Applications, Web Applications, etc) and provides REQUEST-RESPONSE, ROUTER-DEALER (Asynchronous) and PUBLISH-SUBSCRIBE communication modes. It should also be platform-agnostic (Should be able to work with Linux-based, Windows-based and VSWorks-based systems).

The other part of the project objective is to develop a set of automation tools to automate the following tasks that are currently handled by the MRI SW-Infra Team,

such that it resolves the fragmentation prevalent in the current system, and would require minimal user interaction so that it can be used by any person. The first task is Service Pack Framework Automation. Service Packs normally comprise of bug-fixes, code improvements and/or new functionality to an existing release. Currently, the process of Service Pack creation is a non-automatic process, involving a sequence of manual steps which can take an entire day to complete. Also, these steps have to be followed separately for each release that the Service Pack is intended for. The new system should be able to automatically generate a Service Pack after performing certain checks and obtaining the required version information from the user. it should take not more than a few hours to complete, and should not be require any user interaction except for providing version information at the beginning. It should also be able to build an existing service pack, as well as generate a report depicting the health and other vital statistics about the Service Pack. The second task involves automating the configuration tests, currently done by Field Service Engineers (FSEs). Typically, MR systems are provided support for at least 10 years, during which some components (For Eg: Coils) may wear out and would need replacement(s), or some new feature may not support one or more components of the MR Machine, and thus may need to be upgraded. Sometimes, the replacement component may conflict with some other system component(s), and thus may lead to system failure. Currently, this problem is overcome by performing regression testing, which may take anywhere from 6 to 8 weeks. This poses serious challenges and is vulnerable to human errors. MR supports more than 800 configurations, while a typical MR system would have around 81 attribute having a range of 215 values and 51 intertwined validation rules. Regression through these many configurations is challenging, lengthy and there is a risk that some values may be skipped. The new system should be able to speed up this process to few hours from the 4-6 weeks it currently takes. It should generate all combination possible for a given set of attribute value range, and then validate them against the validation rules obtained from the validation rules XML file.

Chapter 2

Literature Survey

2.1 Magnetic Resonance Imaging

Magnetic Resonance Imaging is a 3-dimensional imaging technique used to visualize parts of or the whole body and its structure, without penetrating the body or using any kind of radiation. MRI provides high quality images that are rich in detail and can show even the minute changes that occur over a period of time. It is very useful in detecting the structural abnormalities that develop over time due to a disease/disorder, that can prove to be extremely helpful to identify different aspects of the disease/disorder.

Tissue that contains a large amount of hydrogen, which occurs abundantly in the human body in the form of water, produces a bright image, whereas tissue that contains little or no hydrogen (e.g., bone) appears black. The brightness of an MRI image is facilitated by the use of a contrast agent such as gadodiamide, which patients ingest or are injected with prior to the procedure. These agents can improve the quality of images from MRI. For improving the sensitivity, techniques have been developed, such as the use of para-hydrogen, a form of hydrogen with unique molecular spin properties that are highly sensitive to magnetic fields.

2.1.1 History

Magnetic Resonance Imaging or MRI is used to produce detailed anatomical images of the human body (or any other body for that matter). It is a non-invasive (as it makes no contact with the body) imaging technology, and unlike other technologies like X-Rays, it doesn't produce radiation. Apart from imaging (MRI images are used extensively to detect/diagnose diseases, injuries, tissue damage, blood flow detection, cancerous cells/tumor, etc), it can also be used for MR therapy (For eg: MR therapy can be used to treat tumors, in place of harmful and painful chemotherapy).

In 1946, a researcher by the name Felix Bloch, in his Nobel Prize winning paper, proposed rather new properties for the nucleus. He stated that the nucleus behaves like a positively charged magnet, due to the presence of protons. He also showed that charged particles like protons spin around its own axis has a magnetic field (The magnetic momentum). This is known as the Bloch Equations. During the 1950s, this equation was verified experimentally, and in 1960 Nuclear Magnetic Resonance (NMR) spectrometers were developed to analyze the molecular configuration of materials based on their NMR spectrum. During the late 1960s, Raymond Damadian discovered that malignant tissue had different NMR parameters than normal tissue and thus tissue characterization is possible; and in 1974, he produced the first ever NMR image of a rat tumor on the basis of this theory. The first super-conducting NMR Scanner called "The Indomitable" was constructed in 1977, which took almost 5 hours to produce the first ever image of the human body. The name Nuclear Magnetic Resonance (NMR) was changed into Magnetic Resonance Imaging (MRI) as the word "nuclear" would not have gone down well with the normal public. Modern MRI machines are capable of a wide range of applications, such as Functional Imaging, Perfusion Diffusion Scanning, MR Angiography, etc.

R&D over the years in Magnetic Resonance Imaging has brought about great advancements in the field. Sensitive imaging techniques have been developed that can image specific portions of tissues and can identify specific properties based on the

field strength, contrast agent, its quality, frequency of RF coil, etc. Techniques such as Diffusion MRI and Functional MRI have been developed. MR Angiography can be used to image the blood flow blood inside a part or the whole body.

2.1.2 Concept

Basically, MRI uses the phenomenon of excitation and directional change of rotational axis in protons when manipulated by magnetic fields. Basically, MRI machines contain a very strong magnet (With strengths ranging from 1.5 Tesla to even 7 Tesla), to which the body is exposed. As we know, a water molecule contains two hydrogen atoms, which are positively charged. As our body contains more than 70% water; when exposed to the strong magnetic field, these atoms (or protons) align themselves to the direction of the strong magnetic field. let us consider this direction as the Z-axis. Depending on the part of the body to scan, magnetic field is also applied along the X and Y axis. This secondary magnetic field's strength and orientation depends on the type of element whose presence has to be detected. A radio frequency signal is then pulsed through the RF coil, wrapped around the part of the body to be scanned, and the protons are thus stimulated, an they spin out of equilibrium, the direction depending on the strength of the RF signal. When the RF field is turned off, protons realign themselves to their original position, an emit the excess energy as radiation (Photons). This is then captured by the MRI sensors. The time taken by the protons to realign back to the magnetic field and the amount of energy emitted depends on the strength of the signal applied as well as the type of molecule. Thus, different compositions of the body are identified. Often, some contrast agent(s) (for eg : Gadolinium) is/are given to the patient to change the behaviour of the protons, so as to generate different kinds of images.

Typically, during an MRI scan, the patient is made to lie inside a hollow cylindrical magnet (as shown in the figure). This magnet is made using superconducting coils, through which electric current is passed. Due to Faraday's law, a magnetic field is

generated around the coil in the perpendicular direction. This superconductivity is obtained by lowering the temperature of the coil to an extremely low temperature (depends on the superconducting material) by keeping the machine submerged in liquid helium. Helium is used for supercooling as it is inert so causes no interference to the electric current flowing inside the coil. The machine is places inside a cage (Faraday's cage) so that no signal escapes outside or no outer signal cones inside to produce noise. Thus, the patient is exposed to a very powerful magnetic field (1.5 Tesla, 3 Tesla or even 7 Tesla). As each element (nucleus) contains different charge, their resonating frequency for the steady magnetic field is different. Taking advantage of this, another magnetic field is applied on the body at a different angle (orientation), and is turned on and off repeatedly at different frequencies. The atoms that resonate at that applied frequency line up to the second magnetic field and when it is turned off, they go back to the original alignment, and in the process emit the signals, which are then picked up and recorded. Finally, these values are used to construct an image. MR scanners work especially well for non-bony parts or soft tissues of the body. In contrast to Computed Tomography (CT), MR scanners don't use any ionizing radiation (Like X-Rays do). This, they are better suited to scan brain, nervous systems, blood veins, spinal cord, muscles and ligaments, etc than CT Scanners r X-Rays. In case of brain, no other method produces as detailed images as MR scanners.For eg: MRI can be used to determine which areas of the brain are active (by identifying Oxygen consumption) while performing various cognitive tasks. MRI is thus widely used for advanced brain research and to assess neurological status/risks. Since MRI images lack in visualizing bone structures (due to scarce amount of bones there), intracranial and intraspinal scan images are found to be of excellent quality.

2.1.3 Advantages

- It is a non-invasive (as it makes no contact with the body) imaging technology, and unlike other technologies like X-Rays, it doesn't produce any kind of

radiation.

- MR image qualities are far superior to that of the other imaging methodologies:
  - X-Ray images are flat and gray in structure. Moreover, they have a very poor resolution, which can be improved only by administering some "contrast medium" to the patient. the contrast mediums are usually based on barium or iodine, but they have side effects on the body, and thus for multiple imaging they are not feasible.
  - CT Scanners produce images having a lot more contrast, but still cannot match the quality and diversity of MR Images.
- In addition to having an excellent contrast resolution, which provide the ability to detect minute contrast differences in soft tissues, MRI Images provide the possibility the generate multiple images in any plane/section, which is not possible using X-Ray or CT.
- However, spatial resolution of X-Ray images is better as compared to MRI. Thus, in general, X-Ray and CT Scans are useful when bone structures are to be imaged, whereas MRI is useful for soft tissue lesions, blood stream detection, tumor detection, etc.
- In recent years, use of MR for therapy is being researched. MR is being increasingly used in therapy for treatment of cancer, uterine fibroids, etc.
- MR is being increasingly used for tumors. Instead of chemotherapy, which is too painful and has a lot of side effects, MR is rather safe, painless and has almost no side effects (though that depends on the patient).

Figure 2.1: Brain Scan For Ingenia 3.0 Tesla MR Machine

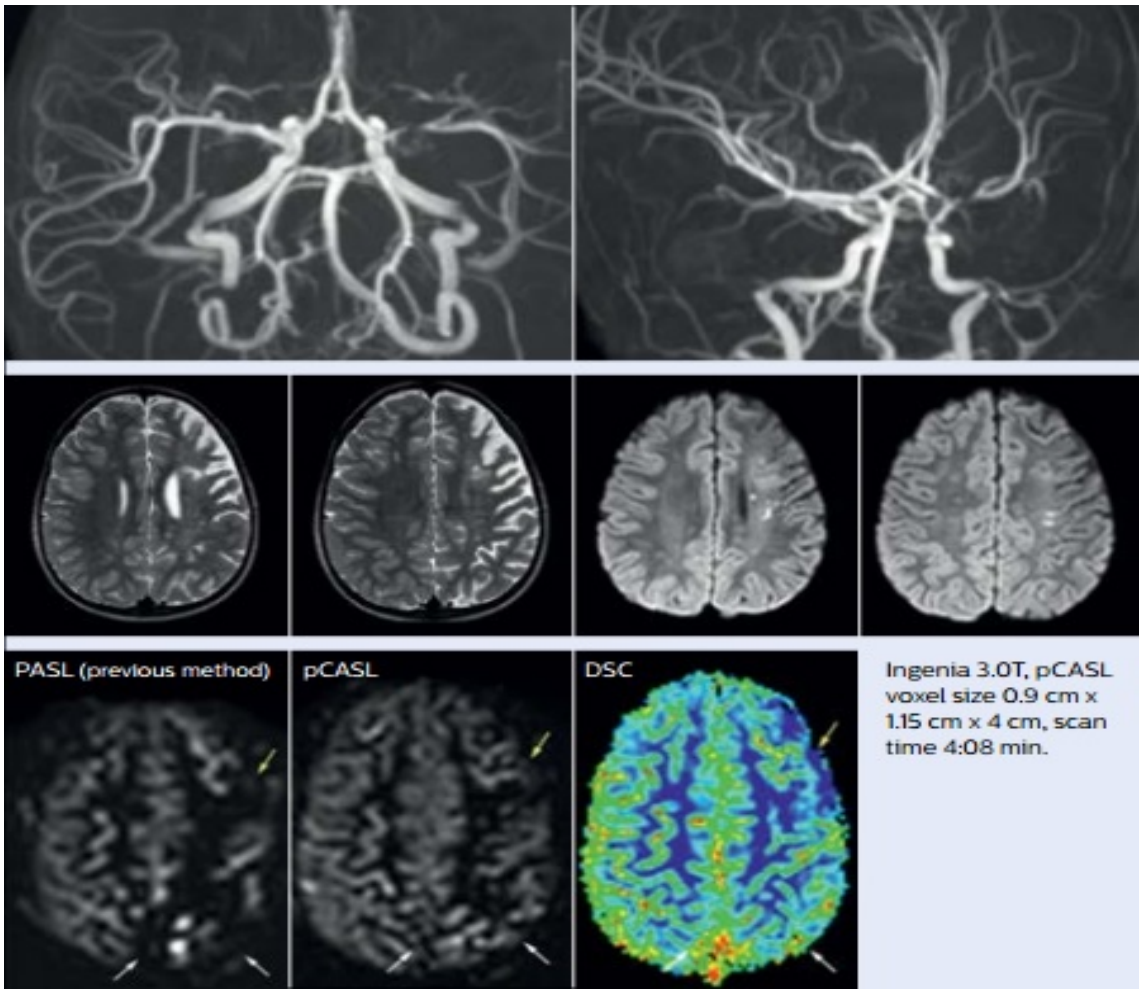


Figure 2.2: Brain Scan Samples For Ingenia 3.0 Tesla MR Machine

2.1.4 Drawbacks

- The biggest drawback of MR systems is the strong Magnetic Field. MR it employs a strong magnetic field, which is capable of flinging an entire wheelchair across the room. Thus a lot of caution and safety measures should be followed while using an MRI machine. There have been cases of death due to negligence on the part of the machine operator in taking precautionary measures or failing to make sure the patient has removed everything from his clothes/body that can be affected by the magnetic field (For eg: Coins, Keys, etc).

- Sometimes implants may stop working due to the impact of the magnetic field. There has been an instance of a pacemaker stopping due to the impact of the string magnetic field of an MRI Machine. Thus, patients with implants, particularly the ones containing iron or ferromagnetic materials (For eg : pacemakers, deep brain simulators, cochlear implants, capsule endoscopy capsules,etc) should not enter any MRI machine.

- MRI machines normally make loud noise while scanning generated due to the residual signals. Its intensity can sometimes go up to as much as 120 Decibels.

- People with claustrophobia cannot to tolerate the long scan times inside the machine.

- Some patients may experience a twitching sensation sometimes due to the rapidly changing magnetic fields inside the machine.

- Some patients experiencing renal failure and who require dialysis are susceptible to Nephrogenic Systemic Fibrosis, which is thought to be caused by some of the gadolinium-containing materials used as contrasting agents, like gadodiamide, though is has not been fully proven to cause the disease.

- Pregnant women are advised to avoid MRI scans as the contrasting agents used may enter the still-forming and delicate bloodstream of the foetus.

2.2 Message Queues

Communication between processes / applications can be carried out using several approaches, as listed below. All the methods described above have their own advantages and disadvantages, depending on when and where they are used.

a. Semaphores

   Includes the use of one (mutex) or more bits (semaphore) to indicate a state or permissions. Typically used by Operating Systems' internal communication.

b. Shared Memory

   Includes the use of a shared memory location between nodes. It has concurrency and scalability issues.

c. Pipes / Named Pipes

   Piping is a process where one the output of a process is connected to the input of another. Again, has scalability issues.

d. Message Passing

   Message Passing is a system where a separate process(es) handles the communication. The process is responsible to connect to the recipient and deliver the message / data.

e. Files / Memory Mapped Files

   Here a file stored locally or on the network or a file mapped to the RAM / virtual memory is used for communication. It has concurrency issues.

f. Sockets

   Here, sockets are used to send data as streams. The problem here is that message boundaries are not defined, and data is sent as a continuous stream.

g. Message Queues

Message Queues send data with well-defined message boundaries, and provide functions above just the sending of data.

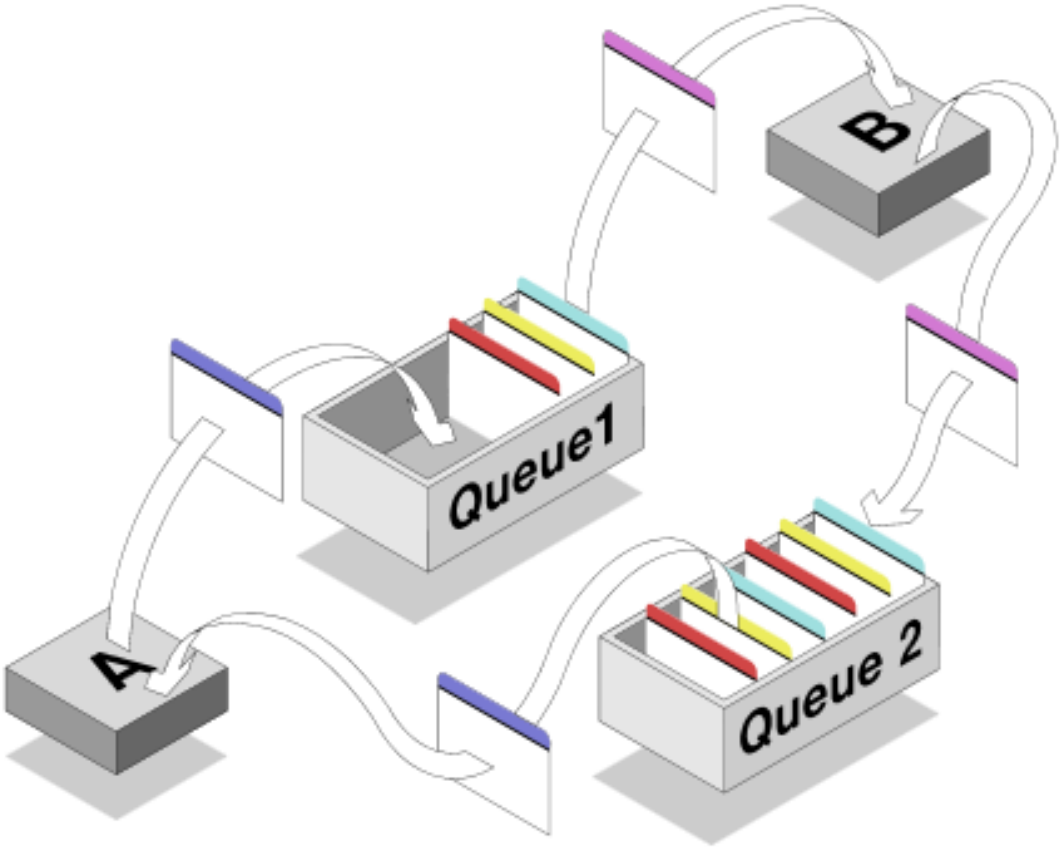


Figure 2.3: The Messaging Queue Concept

We require a message-oriented middleware that caters to a distributed network of MRI systems, which has to be fast, fault-tolerant, highly scalable and capable of tolerating sudden spurts of data.

Messaging Queues, as the name suggests, provide a way for communication between processes, applications or between threads. This communication is characterised by the use of queues for storing outbound/inbound messages, and the use of asynchronous communication protocol for sending/receiving messages.

Message Queues have several advantages over other methods for carrying out communication between processes and/or applications, including the ability to be deployed over a distributed environment. They are typically more robust, fault-tolerant and scalable than other methods, for eg. the client-server architecture.

The main advantage of a messaging queue lies in the use of queues to store messages at the sender's as well as the receiver's end. This ensures that the message(s) can wait for some time if the recipient is not ready, is busy or is unable to cope up with the sender's speed. Again, the use of persistent storage for queues means that the sender and receiver do not need to connect to the network at the same time to communicate, thus avoiding the problems that arise with the use of intermittent connectivity. This also provides a mechanism to recover if the receiver fails, with no data loss, as the receiver can consume the messages once it restarts.

Messaging Queue Systems are distributed applications which take a request to send data from an application / process and send it thorough the corresponding socket, while the receiver, which is listening to the same socket, receives it. Over and above this basic functionality, various features such as routing, support for arbitrary messaging patterns, support for arbitrary language bindings, etc can be implemented on it.

Message Queues may also provide the following characteristics, depending on its architecture and use:

a. Persistence

b. Transaction Support / Distributed Transaction Support

c. Delivery Semantics (once and only once / at most one / at least one)

d. Multiple Retry Policies (linear delay, exponential back-off, etc.)

e. Administration Capabilities

f. Message Retention / Expiration Policy

g. Message Sequencing / Priority Policy

2.2.1 Message Queues Classification

Message Queues can be broadly classified as the ones using a separate broker (Brokered MQs), the ones working without a dedicated broker (Brokerless MQs) and the ones that distribute several brokers across the network (Distributed Brokered MQs).

Brokered Message Queues

Brokered Message Queues use a dedicated broker to perform its tasks. The broker can also be used to perform other tasks such as routing, load-balancing, support for complex message patterns, centralized management, etc.
Brokers allow the system to be more tunable and powerful as more advanced features can be incorporated into the system. Also, the system would be centrally manageable and highly resistant to failure at the receiver's side. They also have a uniform latency behavior, thus the ratio of sender-latency to receiver-latency is almost near to 1.
The drawback of this kind of messaging system is that it introduces a lot of non-message traffic into the network, slowing it down. Also, as each of the features introduce their own latency onto the system, they increase the latency thus degrading the performance. Also, as all messages have to pass through the broker, it creates a bottleneck onto the system at the broker's node, and also introduces a single point of failure.

Broker-less Message Queues

Here, a dedicated broker is not used. Broker-less Message Queues rely on peer-to-peer communication and the underlying network to deliver the messages.
As there is no extra processing on the messages, the throughput is high, as delay is minimal. Along with being lightweight and less complex to implement, these systems provide high performance. Ideal for systems requiring low-latency and high transactional rate.
The drawback is that these are simple implementations, with no advanced features, no predefined message structures, nor any reliability features. These is also a visible difference between sender-latency and receiver-latency (non-uniform latency behavior). Another drawback is the worsened manageability of the system.

Distributed Broker Message Queues

Here, multiple brokers are used at different points in the network. It tries to include the advantages of the single-brokered messaging queues and mitigate their drawbacks.
There is better processing of messages, the option to use advances features like messaging patterns, while having multiple points of failures and a reduced bottleneck (If one broker is busy, another one can take up its work).
The drawback is that they still introduce a lot of non-message traffic into the network, and the latency associated with each of the advanced features incorporated into these brokers.

Chapter 3

Implementation Methodology

This chapter contains the implementation methodology, divided into two sections. The first section contains the implementation details of the Messaging Framework, while the section described the implementation methodology for the tools developed.

3.1 Messaging Framework

The basic aim of the "Messaging Framework" is to provide a distributed middleware for MR applications to communicate with each other. This communication may occur within the system as well as with other systems / servers over the network. For developing the messaging framework, various current implementations were studied, samples were made and their performance was analyzed to come up with a model for the new system.

Current System

The existing system in place for communication between processes is an archaic system for message passing and IPC, which uses native windows sockets to

communicate via TCP protocol. It has a certain set of disadvantages. The first one is that by using native windows sockets, it is not platform-independent, and cannot support other platforms typically used in MRI such as VxWorks or Linux. Also, it provides limited support for synchronous communication, and no publish-subscribe mechanism. Thus, the need for a more modern, algorithmically efficient and highly scalable system which also provides support for different communication modes.

3.1.1 Comparison of Available Message Queue Implementations

In this section, available message queues are compared based on the following characteristics:

a. Important Observations

b. Licensing Details

c. Brokered / Non-brokered

d. Persistent / Non-persistent

e. Throughput characteristics

f. Delay / Latency characteristics

g. Durability / Clustering / Transaction characteristics

h. Failure recovery characteristics

i. External dependencies

j. Development language

k. Noteworthy In-built Functionality Provided

l. Advantages

m. Disadvantages

MSMQ [Microsoft Message Queuing]

a. Bare-basic, simple store-and-forward, so only send-receive functionality
Onus of implementing everything else is on the developer
Arbitrary hard limitations (For eg. Maximum message size is just 4MB)
Can be used with MassTransit or NServiceBus layer to get better functionality performance
Implemented by Microsoft itself since 1999 (latest release for Windows Server 2012 R2)

b. Comes but bundled with Microsoft OS

c. Brokered

d. Persistent, but not by default

e. Decent performance

f. Decent performance

g. No, but can be implemented (For eg. Queue-Transaction can be enabled using SQL-DTS along with MSMQ)

h. Not Available

i. Nothing

j. Microsoft Native Implementation

k. Durable (i.e. Order (of messages) retained), but has to be configured

l. Advantages:

- Bare-basic functionality, easy-to-deploy and durable
- No external runtime-support required (because it comes bundled with windows OS)

m. Disadvantages:

- Only send-receive, everything else has to be implemented by the user; this practical use is challenging
- Arbitrary hard limitations like maximum message size (4MB)

ZeroMQ

a. Broker-less, non-persistent, high concurrency and highly scalable
   Mature implementation of broker-less Message Queue, has a large community (thus can receive timely fixes, is more stable)
   ZeroMQ considered the perfect message-dispatcher
   C++ code (lots of STL libraries re-implemented here)
   Concurrency achieved through Message-passing, so threads dont interfere with each other (one-to-one relationship between the object and the thread). Thus mutex/semaphores not required

b. Open Source (LGPLv3)

c. Borkerless

d. Non-Persistent

e. High throughput (Better sender throughput, relatively less receiver throughput in comparison with nanomsg)

f. Low latency, high efficiency

g. High concurrency, sequencing (optional)

h. Optional

i. No additional Dependency

j. C++

k. N/A

l. Advantages:

   - Speed, Concurrency, Scalability, High Throughput, Low Latency
   - Open Source, has an active community and is widely used

- A mature product

m. Disadvantages:

- Non-persistent
- No message structure
- No advanced features
- Disconnects slow clients

NANOMSG

a. A socket library, can work on a wide range of Operating Systems without extra dependencies
   A fork of ZeroMQ, an improved version
   Incorporates several optimizations over ZeroMQ
   Interactions are modelled as a set of State-Machines, making it Thread-Safe
   NanoCat: Send/Receive data via nanomsg sockets
   ZeroCopy: Zero CPU utilization during copy operation
   Is still young though, not mature enough

b. Open Source (MIT/X11)

c. Borkerless

d. Non-Persistent

e. High throughput (Average in comparison to ZeroMQ, but more uniform)

f. Higher latency (steep curve for lesser number of messages, becomes more uniform as number of packets increase) when compared to ZeroMQ

g. Preservation of message order, several optimizations over ZeroMQ

h. Not Available

i. No additional Dependency

j. C

k. N/A

l. Advantages:

- Almost the same as ZeroMQ, except the high-latency behavior
- POSIX Compliant, cleaner and more optimized code, better error-handling
- Pluggable interface for Transport and Message protocols, so support can be easily added. For eg. Web Sockets
- Interoperable at API and Protocol level, allowing it to be a drop-in replacement
- Other internal optimizations like being Thread-Safe, the use of Radix Tree, ZeroCopy, NanoCat, etc.

m. Disadvantages:

- Same disadvantages as ZeroMQ
- Not Mature enough, not much active community support

RabbitMQ

a. Supports open AMQP specifications
   Is robust, has enterprise resilience and durability, high availability, etc
   Supports a wide array of features such as persistence, routing, sophisticated messaging patterns, load balancing, negative acknowledgement, etc
   Can be coupled with Celery

b. Open Source (Mozilla Public License)

c. Borkered

d. Persistent

e. Average throughput (High throughput when compared to other AMQP MQs)

f. In the AMQP group, has the least latency; but when compared to others (even brokered-ones like NATS), its latency is several orders of magnitude higher

g. Durable, clustering support, AQMP guarantees atomicity when a transaction involves a single queue

h. Resilient, clustering ensures that it survives hardware failure

i. Erlang runtime required

j. Erlang

k. Routing, sophisticated messaging patterns, load balancing, negative acknowledgement, etc

l. Advantages:

- Decent performance
- Lots of useful advanced message-handling features
- Persistent, robust, scalable, durable, high availability, fault tolerant, has in-built crash/failure-recovery mechanisms

m. Disadvantages:

- Message broker needs maintenance and add its overhead (if enabled), overhead of persistence (if enabled)
- AMQP considered Over-Engineered, several features added which are not required
- Average throughput and latency performance
- Cannot handle network partitions well, behaves unexpectedly

ActiveMQ

a. Supports open AMQP specifications

   Java-based, cross-platform compatibility, has a long service and ubiquitous (i.e. widely used)

   MSMQ, RabbitMQ perform better than ActiveMQ in several performance metrics

   Very powerful, has a lot of advanced features (more than RabbitMQ)

b. Open Source (Apache 2.0 License)

c. Borkered

d. Persistent (Pluggable)

e. Average throughput (slightly better than RabbitMQ)

f. High latency (Even higher than RabbitMQ)

g. Durable, clustering support, AQMP guarantees atomicity when a transaction involves a single queue

h. Resilient, clustering ensures that it survives hardware failure

i. Java runtime required

j. Java

k. Highly configurable, Powerful, Routing, sophisticated messaging patterns, load balancing, negative acknowledgement, etc

l. Advantages:

   - Very Powerful
   - Highly configurable (you can configure almost everything)
   - Lots of useful features (more than RabbitMQ)

- Persistent, robust, scalable, durable, high availability, fault tolerant, has in-built crash/failure-recovery mechanisms

m. Disadvantages:

- High Complexity
- Message broker needs maintenance and add its overhead (if enabled), overhead of persistence (if enabled)
- AMQP considered Over-Engineered, several features added which are not required
- Average throughput and latency performance (latency even worse than RabbitMQ)

Apache QPID

a. Supports open AMQP specifications
   Multiple language options available (See the table at the end)
   Highly configurable, supports message priorities

b. Open Source (Apache 2.0 License)

c. Brokered (For C++ & Java both)

d. Persistent (Using a pluggable layer e.g. Apache Derby)

e. Decent Performance

f. Decent Performance

g. Clustering, Transactions supported, Durable

h. Retry logic present in client node

i. No special requirements (Depends on the implementation used)

j. Pure-Java as well as Native-Code (C++) Implementations

k. Routing, sophisticated messaging patterns, load balancing, negative acknowledgement, etc.

l. Advantages:

   - Available in multiple languages / platforms
   - Advantages of AQMP

m. Disadvantages:

   - Disadvantages of AQMP
   - Message loss when broker crashes (no failure mechanism here)

- AMQP considered Over-Engineered, several features added which are not required
- Can be considered a scaled-down multi-language version of ActiveMQ

3.1.2 Implementation

The messaging framework developed is an efficient, highly scalable and high performing system that supports multiple platforms such as Windows, VxWorks and Linux (as it uses the ZeroMQ socket implementation which has implementations in multiple platforms) and multiple development languages. The messaging framework uses messages queues which, as described in the chapter "Literature Survey", use queues for communication. The messaging framework is a broker-less service (on the lines of ZeroMQ). The reason for not using a broker lies in the fact that a broker introduces a performance bottleneck and a single (or multiple, depending on the number of brokers) point of failure, and the benefits of a brokered MQ are not of much use in the MRI context, as the nodes (MR Systems in our context) are homogeneous, have high availability, most of them have high speed Internet connectivity and they can always be modified to suit our requirements. Thus stuff like advanced message pattern support, persistence, etc are not required. Brokerless systems mean low latency and high performance. Although they pose a risk of message loss, the probability of message loss in the current context is very low, and there are mechanisms to deal with it.

Queues provide several benefits, as they provide the possibility for retries. When a message has to be sent, the sending node polls for the availability of the receiver node. If the node is available (i.e. there is a socket connection to that node), the message is sent through the corresponding socket. Else, the message shall remain in the queue for that particular socket, and the sending is retried after the user-defined time-interval. Another aspect of message queues is its asynchronous nature, which is very important in the MR Context. The receiving process may not be available or may be busy when the sender wants to communicate, and queues provide a way to work in that environment.

Based on the above analysis, ZeroMQ was chosen as the message queue because

of certain advantages it has over conventional queues. First, it has a simple bare-basic message queue implementation which provides no overhead and uses efficient algorithms for its tasks. The second point is that it natively sends data as a byte stream, so it can be used to support multiple languages and platforms easily. Another advantage is that it natively provides support for three communication modes, namely REQUEST-REPLY, ROUTER-DEALER and PUBLISH-SUBSCRIBE which are important for our use-case. Thus, the messaging framework is built upon ZeroMQ. MR systems have applications written in C++, C# and Web Applications. ZeroMQ is implemented in C++, and an implementation in C# was built based on its C++ source code. The messaging framework developed also has a web implementation, which uses the HTML5 WebSockets for communication, and supports the three message patterns as described above. The web implementation consists of a server in C#, which provides a layer of abstraction between the message queue and the WebSocket communication. The server uses an open source C# websocket implementation called fleck to communicate with the client. On the other hand, modules are written in AngularJS to handle communication on client side. Currently, the client-side implementation provides only basic REQUEST-RESPONSE, while the 3 communication patterns as described below have to be implemented.

For our messaging framework, the messages are structures into 3 parts, the first part indicates the length of the message, the second part indicates the message type and the third part is the actual message. When messages are to be sent, they are divided into frames, and certain flags aer added to it based on the messaging pattern used. One of them is the "sendmore" flag, which is used to indicate if more frames are arriving. If that flag is set, the messaging framework waits for other frames. When it receives a frame with that flag reset, it combines the frames to reconstruct the original message. The messaging framework provides support for three communication modes as described below. These modes are implemented as a wrapper on the top of the message queue implementation.

REQUEST-REPLY

The Request-Reply mechanism works as it is named. Basically, we can say that it maps to Remote Procedure Call and the classic client/server model. One node sends a request to another node requesting for communication, and the other node sends an acknowledgement, thus implicitly accepting the communication. This form of communication requires synchronisation, which is achieved by introducing a handshaking mechanism and a keep-alive mechanism (using heartbeat messages). Consider a client that wants to communicate with the server, which is listening to the port 5556. For that, the client firsts calls the messaging framework and asks for a connection to the said port. If the port is available (i.e. no other process is using it to communicate with the said server), the client is provided that port to communicate and the client sends its REQUEST through that port, else it shall have to wait for the port to be released. On the other side, the server is listening on the said port for any messages, by binding its REPLY socket to the port 5556. The server waits for a request message in a loop, and responds to it with a REPLY if it wants to establish a connection. The client sends a request and reads the REPLY back from the server. Here, if the server fails, the client won't recover properly. If the server fails while processing a request from the client, the client obviously won't get an answer back. For that matter, if something like this happens, a timeout is set, after which the client shall wait for sometime and try again, or try to connect to another server, if available.

ROUTER-DEALER

The Router-Dealer mechanism is an asynchronous communication pattern which is intended to work for distributed systems. A typical distributed system consists of multiple layers spread over a huge number of systems, and include intermediate aggregation points which route the messages to other components

which in turn may route them again until eventually they reach their destination. In order for a response to make it back to its originating node after hopping through multiple intermediates, the message must contain the address of every hop along the route. Thus, from source to destination, each intermediate ROUTER socket in this ROUTER prepends the sender's identity to the message (this is an optimization to save on bandwidth). When the message is received by the destination, it now has a complete record of every socket that touched the packet on its way to the destination. When the ROUTER socket writes a response, it must prepend all of that routing information to the response so each intermediate node knows how to route the message to the next hop. Each intermediate ROUTER node pops the top routing message part off and uses the new top of the stack to route the message. This model is highly resistant to failures, because if the receiver is down when the client has sent the message, the messages wait in the buffer (queue) and the receiver will be able to consume those messages when it comes up.

PUBLISH-SUBSCRIBE

PUBLISH-SUBSCRIBE is a complicated communication pattern, wherein the publisher is able to send multicast messages, which should be received only by the subscribers to the said publisher. The PUBLISH-SUBSCRIBE communication is an asynchronous form of communication, wherein starting the publisher and the subscriber are independent of each other, thus even if there are no subscribers to it or one or more subscribers have failed, the publisher should go on publishing without having the knowledge of its subscribers. Also, a subscriber should be able to connect to multiple publishers, and the messages received by it shall be interleaved, so as to avoid a single publisher drowning out the others. Here, the recipients of messages (subscribers) don't talk back to the senders, thus reducing "back-chatter", making it more lightweight. This also means

that the messages are queued and the filtering occurs at the subscriber's end and messages can be lost if it's queue overflows. Here, Message Matching is used identify the publisher-subscriber mapping (i.e. if a publisher publishes something, the messaging framework should know the corresponding subscribers). Matching messages to the corresponding subscribers is a typical bottleneck in a message-oriented middleware (our message queue here). For that purpose, the Inverted Bitmap algorithm is used. The inverted bitmap technique thus works by pre-indexing a set of searchable items so that a search request can be resolved with a minimal number of operations. It is efficient if and only if the set of searchable items is relatively stable with respect to the number of search requests. Otherwise the cost of re-indexing is excessive. In our case, When we apply the inverted bitmap technique to message matching, we may be confused into thinking that the message is the "searchable item". This seems logical except that message matching requests are relatively stable with respect to messages. So, the roles are inverted such that the "searchable item" (i.e. the "subscription") instead is the matching request, while the "search request" is the message published. The indexing process thus should work as follows:

- We number each match request from 0 upwards
- We analyse each match request to give a set of criteria tuples
- We store the criteria tuples in a table indexed by name and value
- For each criteria tuple, we store a long bitmap representing each match request that asks for it

The message matching process works as follows:

- We analyse the message to give a set of criteria tuples
- We look up each tuple in the table, giving a set of bitmaps

- We accumulate the bitmaps to give a final result bitmap
- Each 1 bit in the result bitmap represents a matching subscription

For example, consider the following topics:
forex, forex.gbp, forex.eur, forex.usd, trade, trade.usd, trade.jpy

And the following subscriptions:
0 = "forex.*" : matches forex, forex.gbp, forex.eur, forex.usd
1 = "*.usd" : matches forex.usd, trade.usd
2 = "*.eur" : matches forex.eur
3 = "*" : matches forex, trade

Thus, when we index the matches for each subscription we get these bitmaps:

Criteria	0	1	2	3
forex	1	0	0	1
forex.gbp	1	0	0	0
forex.eur	1	0	1	0
forex.usd	1	1	0	0
trade	0	0	0	1
trade.usd	0	1	0	0
trade.jpy	0	0	0	0

Figure 3.1: Bitmap mappings for the above example [2]

Thus, for the following messages, the scenario is explained below:
Message A - "forex.eur" : 1 0 1 0 = Subscriptions 0, 2
Message B - "$forex$" : 1001 = $Subscriptions 0, 3$
$MessageC -$ "$trade.jpy$" : 0000 = $Nosubscription$

The Inverted Bitmap Algorithm is highly efficient as compared to the traditional "trie" used by ZeroMQ. A "trie" is basically a radix tree, and for each message

matching, the tree is parsed to get the subscription information. While when using inverted bitmap, the cost is reduced considerably as just the bits have to be matched. On the other hand, when a new publisher or subscriber joins or if there is any change in the bitmap table, the cost of re-populating the bitmap is high as compared to the cost of just moving around nodes in a "trie". For our case, the inverted bitmap system proves to be far more efficient than the trie system.

3.1.3 Performance

Sl no.	Size of message	Number of messages	Average time(Round Trip time in ms)
1	10 KB	500	0.64
2	10 KB	1000	0.526
3	10 KB	5000	0.389
4	10 KB	10000	0.4542
5	10 KB	50000	0.4734
6	10 KB	100000	0.50736
7	2MB	100	30.61
8	2MB	500	33.128
9	2MB	1000	33.454
10	2MB	5000	33.0218
11	2MB	10000	33.0332
12	2MB	50000	33.291
13	4MB	100	65.65
14	4MB	500	69.158
15	4MB	1000	69.966
16	4MB	5000	70.2234
17	4MB	10000	69.425
18	4MB	50000	
19	8MB	100	114.08
20	8MB	500	126.978
21	8MB	1000	128.295
22	8MB	5000	130.5144
23	16MB	100	250.4
24	16MB	500	261.9
25	16MB	1000	267.269
26	16MB	5000	262.8982
27	16MB	10000	
28	32MB	100	483.79
29	100MB	10	1314
30	100MB	20	1380
31	100MB	20	1380

Figure 3.2: Performance for Different Message Size & Number of Message

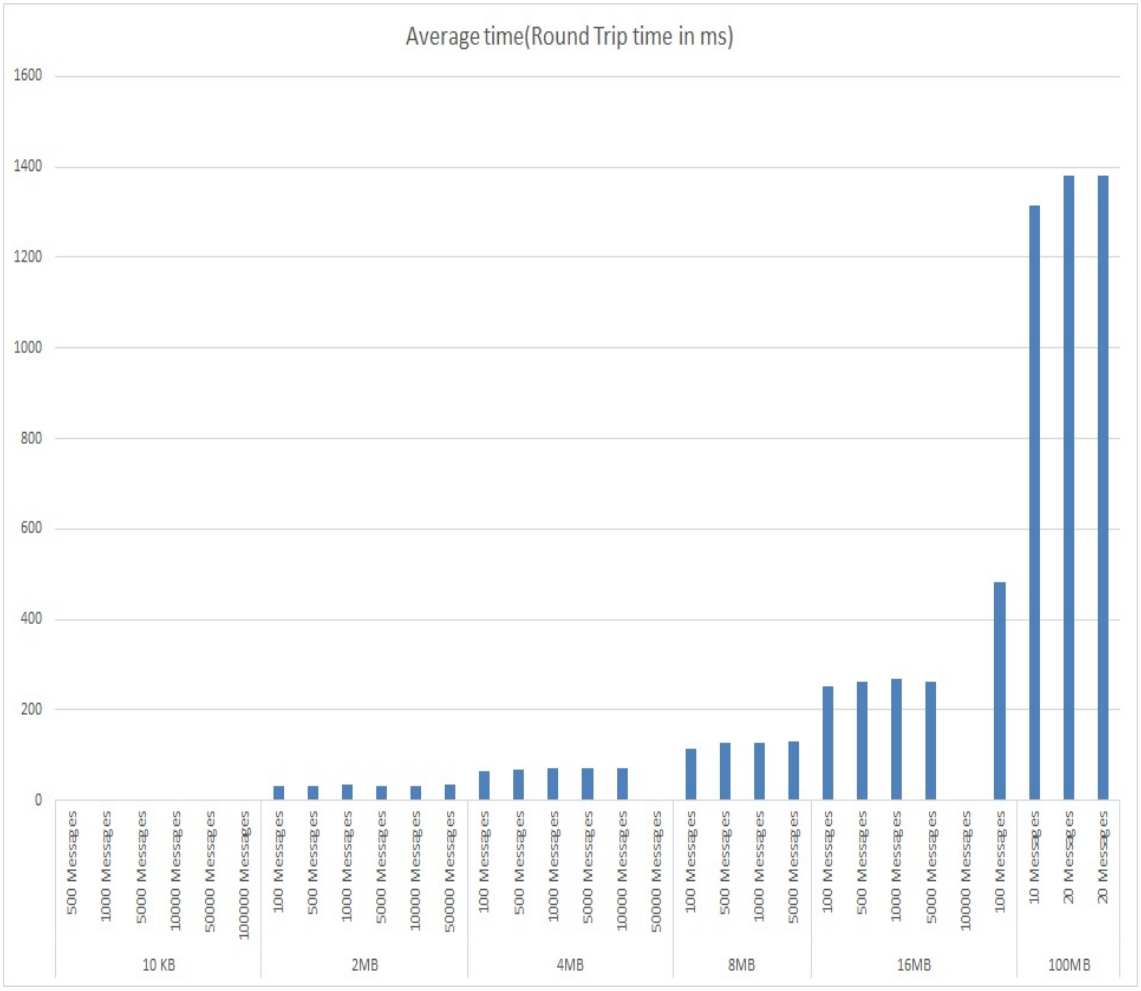


Figure 3.3: Performance for Different Message Size & Number of Message

3.2 Automation Tools

3.2.1 Common UI

As described in the problem statement, the implementation consists of a common framework, that can performed the required tasks. A common UI has been created using Windows Presentation Foundation (WPF). Its package diagram is shown in Figure 3.2.1. It has been developed to be as generic as possible, so that future tools can be easily integrated into the same UI. It provides interfaces for the tool/applications to implement. Through these interfaces, the structure of the UI is maintained, and their DLLs are easily identified through "Reflection".

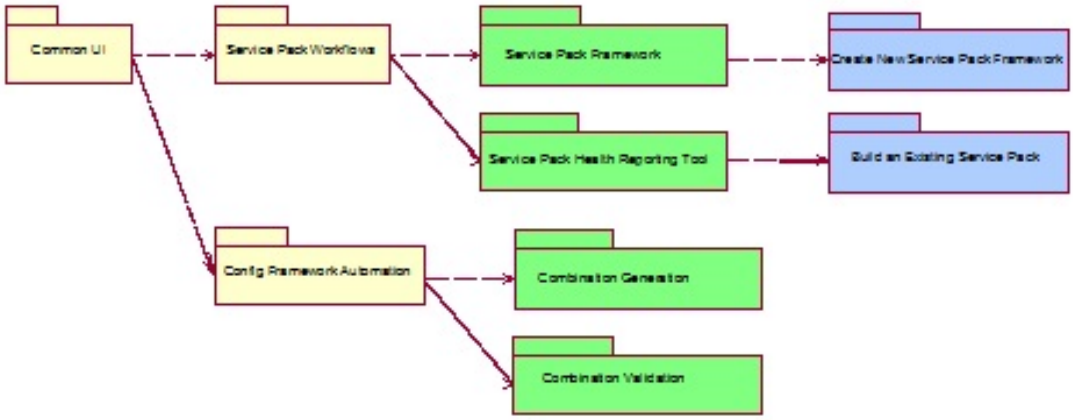


Figure 3.4: Common UI - Package Diagram

The CommonUI has the following characteristics:

- When the UI is launched, it shall scan the "source path" (the path where the required DLLs are present) and, through reflection, shall load the tasks ending with "_task.dll". Is another instance is running from the same stream, a warning is displayed and the UI exits. These are shown in As shown in Figure 3.2.1 and Figure 3.2.1 respectively.

- The UI consists of an interface dll, which contains interfaces for the Pages as well as Tasks to be included in the UI, and a commonUI interface, which consists of definitions for common UI elements such as Radio Buttons, Check Boxes, etc.

- Each task is build as a library (thus generating a dll) and it inherits from the interfaces class. The workflow of each task would typically include taking inputs, specifying file paths, execution page and results page, although it varies from task to task. Snapshots of some pages are shown in Figure 3.2.1 through to Figure 3.2.1.

- The interfaces dll contains an "ISession" interface, which is implemented to get a session up and running for the application instance. The starting pages take the required input from the user in the form of RadioButtons, CheckBoxes, TextFields, etc. These values are added to the session.K

- The most important part is the execution page. Here, the C++ program(s) and the Perl script(s) that is/are to be executed are started as background threads, and the required arguments are passed by taking values from the session. The Standard Input, Output and Error are redirected to the main thread through the "EventDispatcher", i.e. whenever the output or error buffer change, an event is generated, which updates the output string. This string is bind-ed to the TextBlock through "DataBinding", and thus the output "TextBlock" is updated. This ensures that the execution viewof these programs is replicated inside the UI.

- At the end, the output page displays the summary of the execution. If the execution was successful, it would also show hyperlinks for the output file generated. For SP Framework, it shall be an RSI, for Health Reporting it shall be an HTML, etc.

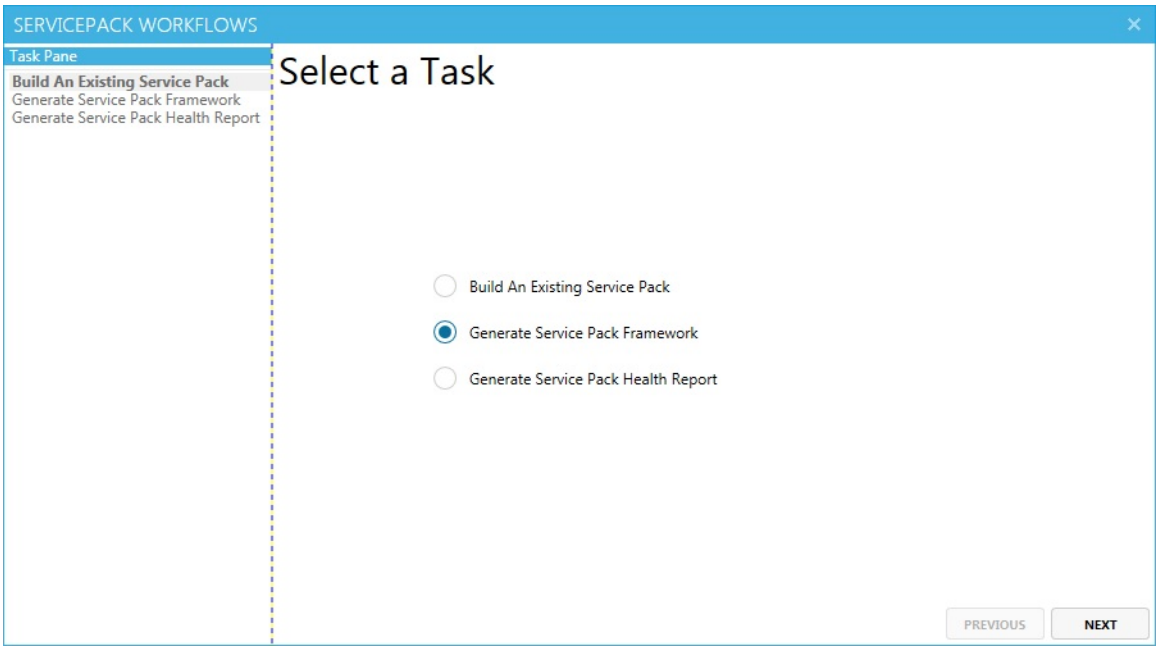


Figure 3.5: Common UI - Main View

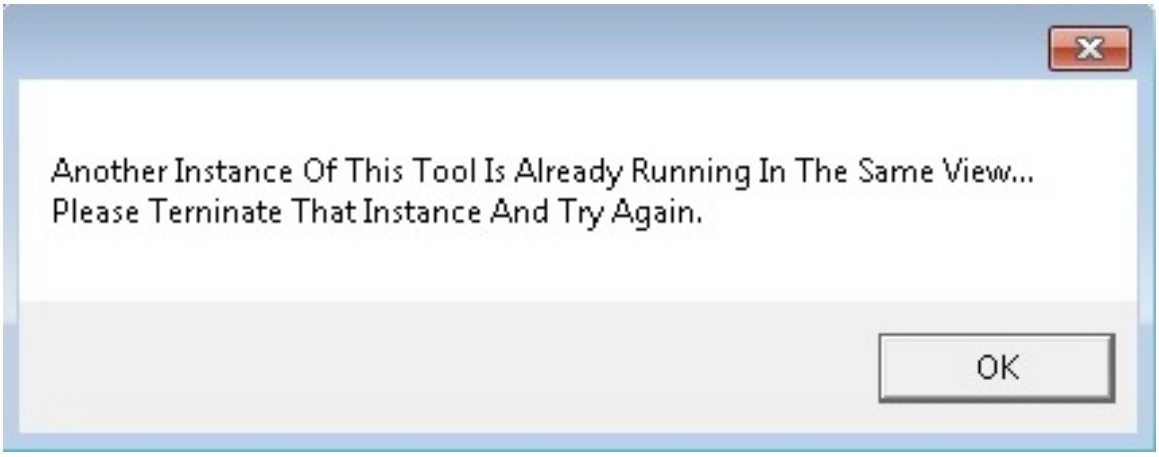


Figure 3.6: Common UI - Duplicate Instance Alert

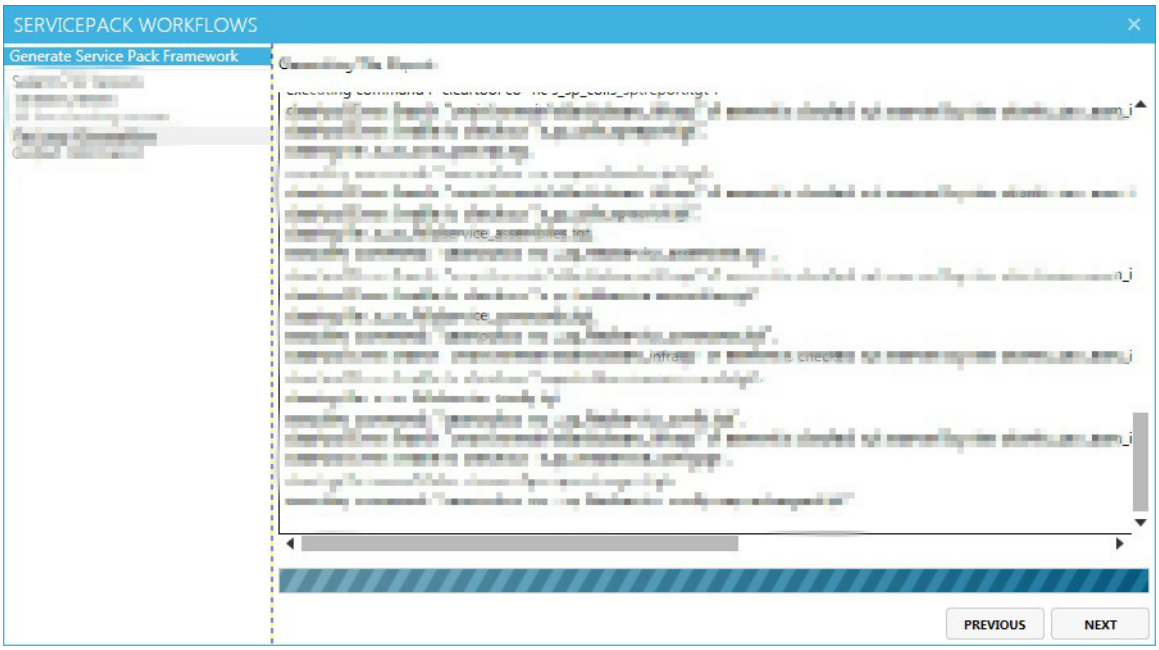


Figure 3.7: Common UI - Process Execution View

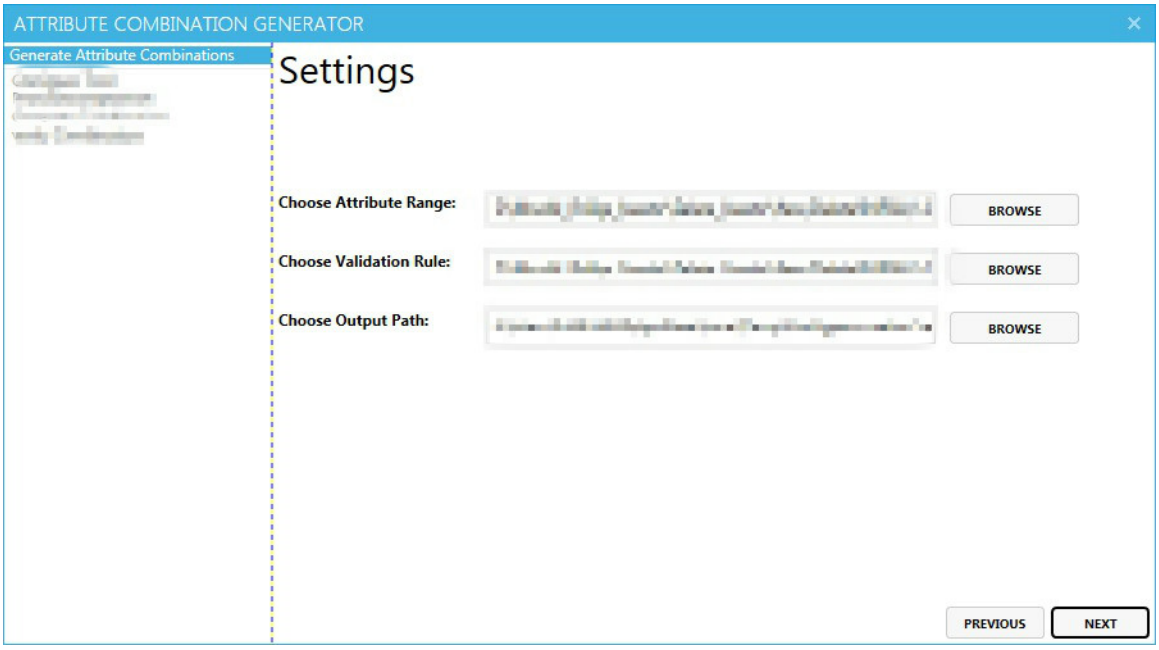


Figure 3.8: Common UI - File Selection View

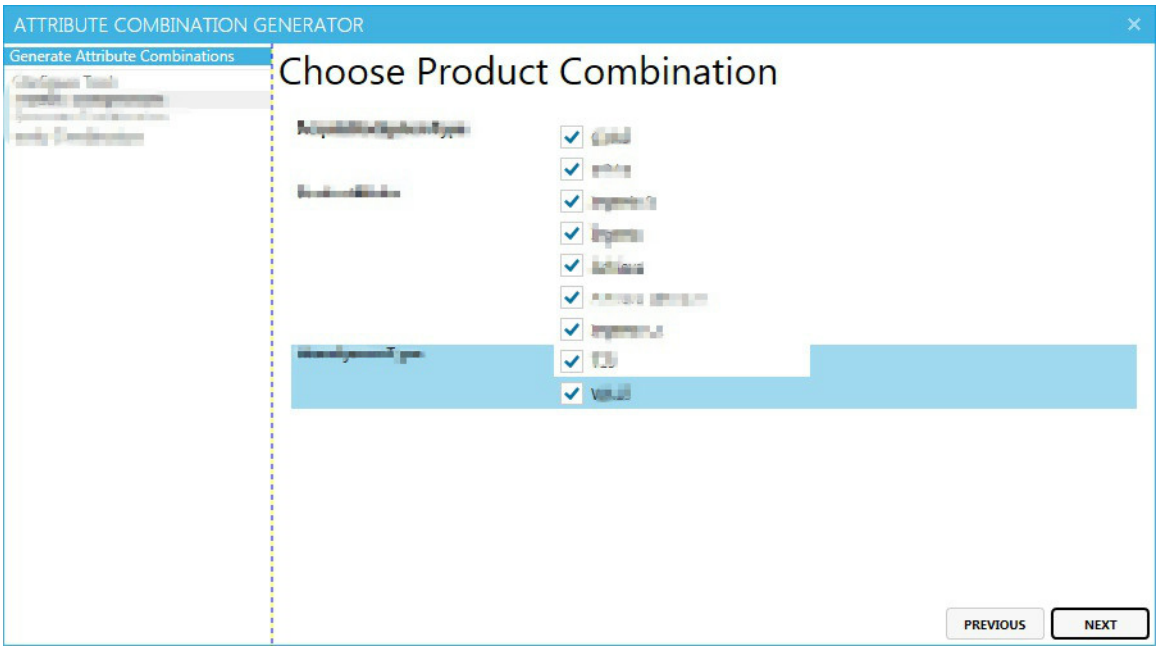


Figure 3.9: Common UI - Attribute Selection View

Figure 3.10: Common UI - Results View (Success)

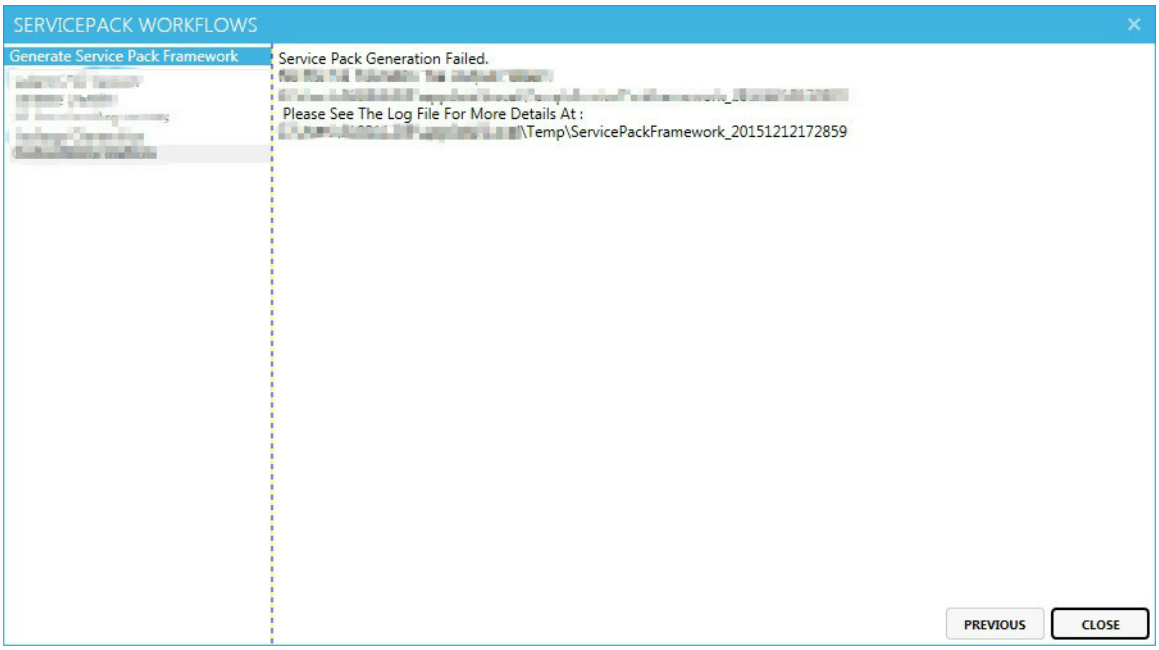


Figure 3.11: Common UI - Results View (Failure)

The UI implementation for each workflow (implemented as a Class Library, i.e. DLL) is described below:

- Generate Service Pack Framework : This generates a new Service Pack Framework. The input consists of only the version information.
- Build An Existing Service Pack : It builds the current system and creates the RSI for an existing Service Pack which has been already deployed. Again, the input consists of only the version information.
- Generate Health Report : It examines and generates a report consisting of vital statistics for the selected release.
- Generate Combinations : For the Configuration Automation, this tool generates the combinations in-memory and then validates the combinations generated.

3.2.2 Service Pack Framework Automation & Health Reporting Tool

The Service Pack Framework, as the name suggests, is used to generate Service Packs, targeted to specific version(s) of specific release(s). In contrast to patches, which provide required minor improvements, bug-fixes and/or system-critical upgrades, Service Packs are created when there are major modifications in software, new features are introduced and/or a major upgrade is required. Service Packs are more generic, and usually target more than one version.

Current Way of Working

The current way of working of SP Framework creation is to perform the following set tasks manually. These tasks are mundane, repetitive and time-consuming. Moreover, it requires a lot of user interaction. The set of tasks vary from release-to-release, and thus have to be repeated for each release. An overview of the steps taken for Service Pack Framework creation is described below:

- The source consists of dedicated building blocks. These blocks contain the source files that are required to be deployed/updated on the machine (DLLs, EXEs, PDBs, etc).
- Whenever a change is made to file(s) in a building block, or new file(s) are added, corresponding targeting is done.
- During Service Pack creation, the configuration file(s) is/are parsed and all the target files are identified. These files are then parsed to get a list of all files along with their current and target paths.
- The GUID, Product Version other values are calculated, current version information is populated and updated to reflect the changes, correspond-

ing system configuration files are checked and updated to reflect the new changes.

- The ISM file is manually updated.
- The Service Pack files are cleared prior to the generation of a new Service Pack Framework.
- The User Documents are generated based upon the list of files and their specified target directories.
- The Service Pack is then built and an MSI is generated.
- The MSI along with the User Documents and other standard files are packed into an RSI.

These steps are performed manually for each release, which is mundane, repetitive and time-consuming. Moreover, it is prone to human error. Thus, the need of a system that can automate the tool.

Implementation

The entire process of Service Pack creation as described above is automated through a set of Perl scripts, which are then called by the Common UI. The inputs to the main automation script would be the starting and ending version information. For example, if my starting version is 5.1.0.1 and ending version is 5.1.0.2, then the Service Pack shall be applicable to these versions as well as to all the versions in between.

Another aspect of this system is the health reporting tool. The Service Pack Health Reporting Tool scans the entire Service Pack and generates HTML reports depicting the health of the Service Pack. The tool actually reads the configuration file and based in the information in it, it scans all files related to the Service Pack and scans all the targeting done. Based on this information, it checks the actual files to be

included in the SP, their path information, their size, their integrity, etc. The final report is generated as an HTML.

Figure 3.2.2 shows the execution flow of the system.

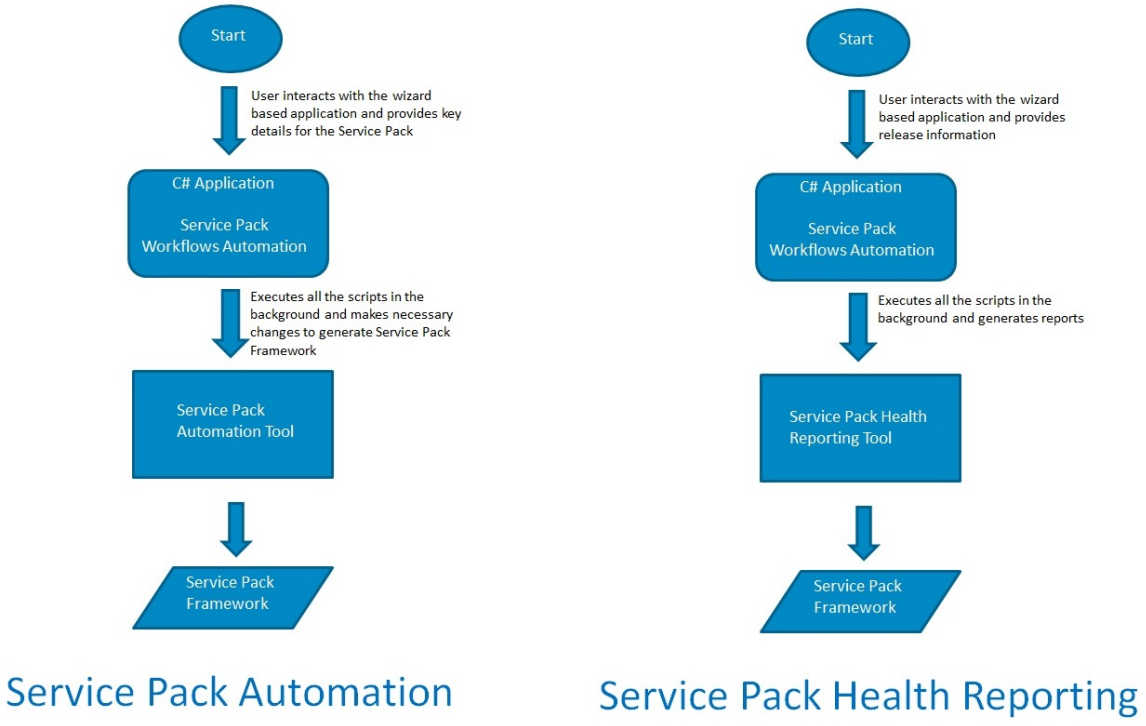


Figure 3.12: Service Pack Framework Automation Execution Flow

The Service Pack Framework creation tool also supports generation of an existing Service Pack. Here, the Service Pack is directly built by looking at the targeting information, and RSI is generated.

Benefits Of The New System

- The new system described above is fully automated, and thus requires no human assistance and has no scope for any error.
- It provides a concise logging of the tasks performed for future reference and for troubleshooting,if needed.

- The health report gives information about configuration file errors, actual file or their path errors, version errors, incorrect targeting, duplicate targeting, etc.
- Finally, it takes a little more than an hour to create a new Service Pack Framework, and just 5-10 minutes to build an existing one. This is a huge gain in efficiency, as the legacy way-of-working would take around 8-10 hours minimum for creating a new Service Pack Framework.

3.2.3 Configuration Framework Automation

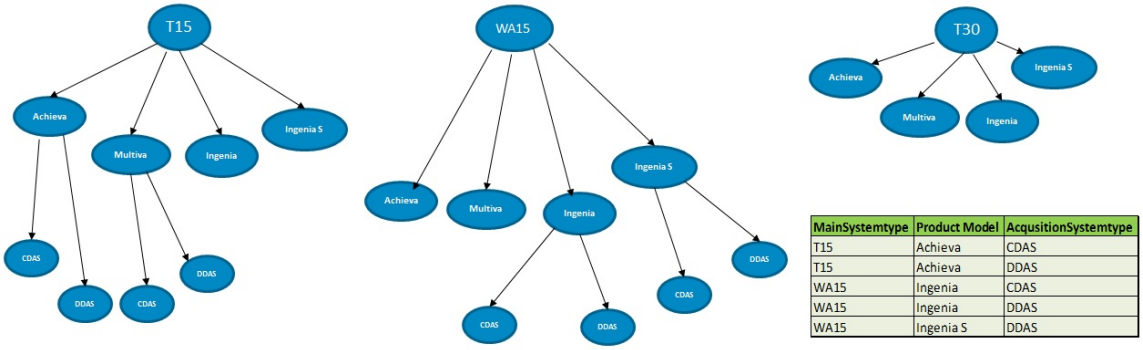
An MR machine consists of a lot of component(s) and configuration(s) (For eg: Coil Type, Magnet Type, etc), having complex inter-dependencies.Sometimes, these need to be changed/replaced, which may result into conflicts with other component(s)/configuration(s), and thus may lead to unexpected system behaviour or component/system failure. Hence, there is a list of validation rules, which need to be conformed to in order to change the particular component/configuration.

Current Way of Working

Currently, this problem is managed by performing regression testing, which may take anywhere from 6 to 8 weeks. This poses serious challenges and is vulnerable to human errors. MR supports more than 800 configurations, while a typical MR system would have around 81 attribute having a range of 215 values and 51 intertwined validation rules. Regression through these many configurations is challenging, lengthy and there is a risk that some values may be skipped (the scope for human error). Thus, the need of a tool that can support an automated test suite intended for fast and comprehensive testing of all the supported system configurations.

Implementation

One of the major problems of CTA is the bulky size of combinations. Because the number of combinations generated can be in millions, they would not fit into the main memory. Thus, the program may crash in some instances. Saving to memory is not preferred, hence a tree-based structure has been created (Figure 3.2.3).



MainSystemtype	Product Model	AcqusitionSystemtype
T15	Achieva	CDAS
T15	Achieva	DDAS
WA15	Ingenia	CDAS
WA15	Ingenia	DDAS
WA15	Ingenia S	DDAS

Figure 3.13: Tree-based Validation

The set of values that, while parsing through the tree, reach the leaf node, are valid, the others are not. This has been implemented in C++, and the attribute range as well as the validation rules are read from their respective xml files, whose files are passed as arguments to the C++ application. This is then hosted into the WPF UI.

The execution steps are:

- The configuration generation tool takes the Configuration Model as input.
- Extracts the information regarding attributes, range and validation rules.
- The combination generation algorithm produces all the configurations that are possible.
- Configurations are the combinations of attribute values that are allowed by the validation rules. These are stores as a CSV file.

- Typically, 237 Billion random combinations are generated, out of which a few are correct combinations.
- The Configuration Validation tool then the Configurations CSV file as input and puts each configuration in the configuration framework under test and checks if they are valid (i.e. the combination of attribute that reached the leaf nodes while parsing).
- Finally, it an HTML report is generated, indicating Pass/Fail against each combination.

Figure 3.2.3 shows the execution flow of the system.

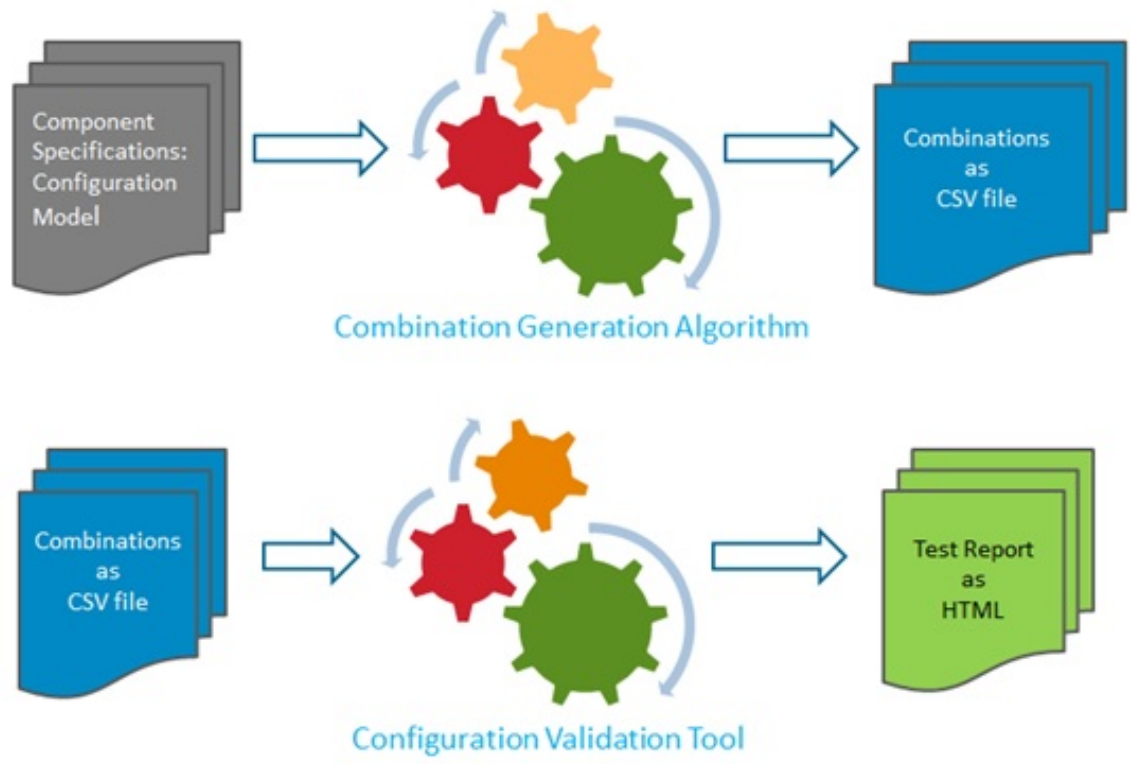


Figure 3.14: Configuration Framework Automation

Benefits Of The New System

- Reduces the testing efforts from 6-8weeks to a mere 15 minutes.
- The optimized configuration generation algorithm creates minimal combinations for maximum coverage.

Chapter 4

Conclusion & Future Work

4.1 Conclusion

The messaging platform, which is being developed is a modern fail-safe and efficient system, which shall improve the response time of various MR process that use it for internal communication. This would have a positive impact on the overall scan time of an MRI machine. For the second part, the system of operations for certain tasks was largely manual, time-consuming, and involved a lot of repetition and was prone to human error. The new systems/tools developed (SP Framework and the CTA Framework) not only fully automates these procedures, but also saves considerable efforts and time. Also, as these systems/tools can be used by any person having basic system knowledge, it considerably reduces the workload of MRI SW-Infra Team. One major factor is that these tools mitigate the scope for human error, while saving human efforts.

4.2 Future Work

The future work for messaging framework includes optimizing the current code so as to improve the performance to the required standards, as well as adding support for the three communication modes, explained in the above chapters, to the web-client as AngularJS modules.

References

[1] Anne Bright. Planning and positioning in mri, 2011.

[2] Pieter Hintjens. ZeroMQ whitepapers, 2016.

[3] Philips. Philips Healthcare systems, 2015.

[4] Marc Schaffrath. Service pack user manual, 2011.

[5] Peter van der Muelen. Introduction to magnetic resonance imaging, 2015.