FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Rule Engine Based Notification System for Health Critical Results Follow-Up

Luis Torres Costa

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Miguel Pimenta Monteiro

March 8, 2018

Rule Engine Based Notification System for Health Critical Results Follow-Up

Luis Torres Costa

Mestrado Integrado em Engenharia Informática e Computação

March 8, 2018

Abstract

Nowadays, with the recent technological breakthroughs, it has become very easy to store large amounts of data of several kinds. Health-care systems are no exception, since it is possible to record whole patients' medical histories, including notes made by health practitioners, prescriptions from appointments, clinical exams and lab tests they have requested as well as the respective results.

However, generally, it is up to the practitioners to decide when they should consult their patients' available information. It means that, a practitioner that has requested an exam, will have to remember later to check if its result is already at his disposition.

This process raises a few problems, especially when the test gives out a critical result. In this case, in order not to endanger the patient's health, the practitioner should have been able to react immediately.

To solve this problem, this dissertation proposes the conception of a notifications system based on an expandable and generic rule engine, allowing the practitioner to create and subscribe to customizable notifications. The engine must first be fed by agents that send meta-data so that, afterwards, they are able to send the data itself. Meta-data dictate the structure, format and even content of the respective data. It's based on this meta-data, that professionals will be able to create rules which will be analyzed together with the incoming records of data and, when there is a match between the both, a notification is launched with its content being customized by the practitioner, and, if needed, including the information within the data records used in the analysis.

The developed system, Rule Engine based Notification System (RENoS), was tested across several aspects, including its scaling ability, meaning, whether the influence caused by the increasing number of data records rendered the system slow or even unusable. To test it, three agents were simulated, that inserted a million documents in the database of three different kinds: person, prescriptions and lab tests. After that, six rules were created that use the previous data in different ways, with varying complexity. Results were positive, with a notification being created in 1,5 seconds on average since its reception in the system, of which only a second corresponds to the analysis itself.

The implemented solution seems to be, then, a viable answer to the critical results notification problem.

Keywords

Medical decision support; Rule engine; Critical results communication

CCS Concepts

• **Applied computing Life and medical sciences** • Applied computing Consumer health • Applied computing Health care information systems

Acknowledgements

This dissertation represents the end of my academic life, therefore there are lot of people I must thank for achieving this feat.

First, I have to thank my whole family, but specially my parents, brother and sister, for giving me this chance, always believing in me and supporting me when I needed.

Secondly, a special thanks to both Professor Miguel Pimenta Monteiro and Engineer Pedro Pinto, for guiding me through this final stage.

I must also thank all my friends, either the ones I already had before starting this journey and those that I only met in the last years, for simply being there.

Finally, I need to thanks all the people I didn't mention specifically, but still heavily influenced my success in this accomplishment.

Thank you!

Luis Torres Costa

"Intelligence is the ability to avoid doing work,
yet getting the work done."

Linus Torvalds

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Abbreviations

AMIA	American Medical Informatics Association
AMQP	Advanced Message Queuing Protocol
ANCR	Alert Notification of Critical Results
API	Application Programming Interface
CR	Critical Results
CRUD	Create, Read, Update and Delete
DB	Database
DBMS	Database Management System
EHR	Electronic Health Record
EMR	Electronic Medical Record
ESB	Enterprise Service Bus
FHIR	Fast Health Interoperability Resources
HIT	Health information technology
HL7	Health Level Seven
ICD	International Classification of Disease
IDE	Integrated Development Environment
IMIA	International Medical Informatics Association
LOINC	Logical Observation Identifiers Names and Codes
npm	Node Package Manager
RENoS	Rule Engine based Notification System

Chapter 1

Introduction

This first, introductory chapter, will present the context in which this dissertation is inserted, the motivation behind it and the problems it's meant to solve, its' goals and purpose, as well as the final expected impacts.

1.1 Context

For a while now, computers have been used to aid health professionals. Health information technology (HIT) can be defined as "the application of computers and technology in health care settings" [Her09]. With it, several areas of informatics have been uncovered, such as Bio, Imaging and Nursing informatics. Some, for instance Medical, Health and Clinical informatics don't even have a consistent meaning and are used interchangeably. Fortunately, associations as the International Medical Informatics Association (IMIA) and the American Medical Informatics Association (AMIA) have made some efforts towards defining standards and recommendations towards a better and more effective usage of informatics by health-care providers [Her09, Fra16]

Fast-forwarding to the present days, the recent technological breakthroughs allow the mass storage and rapid availability of information of all formats and sources. Modern health-care informatics systems keep two kinds of records, Electronic Medical Records (EMR) and Electronic Health Records (EHR) of each patient. While EMRs are records of individual episodes of medical care, an EHR "is a longitudinal electronic record of patient health information generated by one or more encounters(...). Included are patient demographics, progress notes, problems, medications, vital signs, past medical history, immunizations, laboratory data and radiology reports" [HIM]. Besides those two, there's also a Personal Health Record (PHR) that's controlled and maintained by the individual patient.

It makes sense then, that this project was carried out at Glintt. Glintt – Global Intelligent Technologies - is a technology and consultancy company in the Health-care area, with more than

20 years of experience whose solutions are used in more than 200 hospitals and clinics and more than 12.000 pharmacies in the Iberian Peninsula [Gli].

1.2 Motivation

When an health-care professional requests a laboratory or imaging test, its results might be considered critical. Critical results "signify situations which may be life threatening or lead to irreversible damage or harm to the patient. Therefore, one of the main issues of post-analytical activity is represented by the CRs notification" [PSPP17]. For example, a nurse must be notified right away if a test result shows that a patient carries a contagious bacteria and should be put into isolation. However, two scenarios generally happen: 1) either the professional isn't notified and it's up to him to remember to check if the result is already available or 2) the professional is notified by telephone with read-back verification after some period which could be crucial to the patient's health.

1.3 Goals

In this document, a solution is proposed to solve the previous mentioned problem. Putting it simply, the goal is to implement an automated critical results notification system. To get the notifications, the professional should create a rule specifying under which conditions the notification is sent, the destination of these notifications as well as the intended channels through which they are received. Due to indicating a state where a patient's health or even life is at risk, critical results have an urgent nature, thus the quickness of the notification's creation is another goal.

1.4 Document Structure

Apart from this chapter, this document has six others. The next one, chapter 2, defines the domain of intervention, the main problem it aims to solve, and includes some other related projects or solutions to the same issue. Chapter 3 presents the different requirements, use case scenarios and features provided by the proposed solution. The following one, chapter 4, describes the solutions' implemented architecture including its several components and how they interact and communicate with each other. Chapter 5 provides a report on the solution's implementation, including an analysis of some of the tools, technologies and methodologies used comparing them to alternatives when applicable. The next chapter, 6, covers the various tests the solution was submitted to, mentioning how testing was made including its preparation; the aspects and information extrapolated from those tests as well as their importance; and discusses the conclusions taken from the obtained results. The final chapter, 7, starts by summing up all the work done for this project, comparing the proposed with the actually implemented solution, assessing whether the proposed goals were achieved and requirements met, and ends with some points of possible future work and development to improve and add to the project.

1.5 Expected Impacts

By implementing the system described in this document, I hope to provide a new and innovative solution to the problem of communicating critical results in a fast and efficient way, and consequently, if the final product is indeed applied, improve critical results follow-up of patients, ultimately improving their safety and quality of care.

Chapter 2

Critical Results Communication

This chapter includes a literature review, stating the domain of intervention by describing the main problem it aims to solve, and some of the related projects or systems created in order to work out the same issue. Lastly, it presents a tool review, stating chosen tools to be used during the development stage of this project, and why they were chosen instead of other alternatives.

2.1 Domain of Intervention

This project addresses the issues of communicating (particularly, on the receiving end) CRs fast and efficiently enough not to allow additional harm to the patient. More specifically, it would allow doctors, nurses, physicians, clinicians and other health practitioners to subscribe to notifications of a patient's state. This project concerns only in generating the notifications, i.e. it assumes that already exists a solution that collects the patient's data and forwards it to this system. That being said, this document also includes related projects that address both sides of the communication, collecting the data and then generating the notifications.

2.2 Related projects

This section shows very different related projects from several sources, proposed along the last years.

2.2.1 Multiple patient monitoring system for proactive health management

The first project being discussed is a patented system for monitoring a group of patients. This system includes collecting sets of measurements from each patient, calculating control values from said measurements, displaying an overview chart of the patients and, additionally, selecting a patient and sending further instructions. This patent describes a monitoring loop (shown in figure 2.1) with all the steps. For the communication between the several devices, it establishes an

architecture (shown in 2.2) comprised of a Clinic workstation, a Clinic Server, which includes the Main Patient Database, and the patients' sites with the monitoring devices [Bro98].

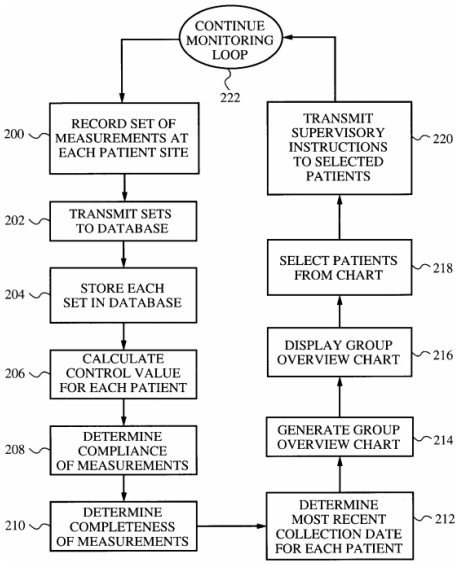

Figure 2.1: Monitoring Loop from [Bro98, Fig. 6]

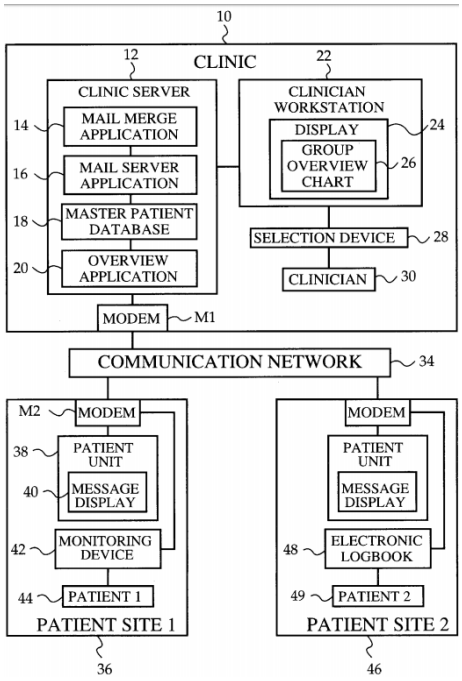

Figure 2.2: System Architecture from [Bro98, Fig. 1]

Although this project doesn't aim particularly to solve the Critical Results communication issue, the monitoring loop and architecture used are very interesting and could be used in such an environment.

2.2.2 Proactive Authenticated Notifications for Health Practitioners

This second project solves the issue by creating an automated telephony server and calling the practitioners' private phone. It uses a Linux server to run the open source telephony server Asterisk. Asterisk keeps checking a shared network folder for new files with the extension "call". This file, put there by the notifying application, contains a PIN for authentication, patient number (which the professional can use in the facility's system to obtain more details), a custom message and possible recipients of the notification. Upon detecting the file, a call is made to the first recipient on the list. If the call isn't answered, the next recipient on the list is called. Otherwise, the recipient is prompted for the PIN. After authentication, through the use of text-to-speech, " the computer reads the notification message". To end this interaction, the user can give some feedback to the system. After hanging up, the system moves it to a "failed" or "done" shared network folder depending on its success. This whole procedure is shown in figure 2.4. According to the authors, this method of two way interaction through phone calls should reduce the calls classified as "missing confirmations" to zero [MSR12].

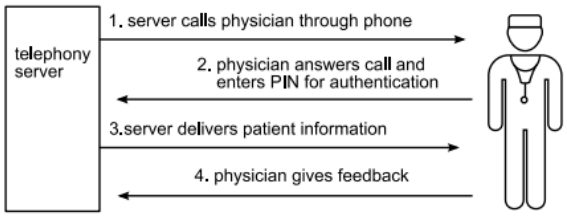

Figure 2.3: Server and physician interaction from [MSR12, Fig. 1]

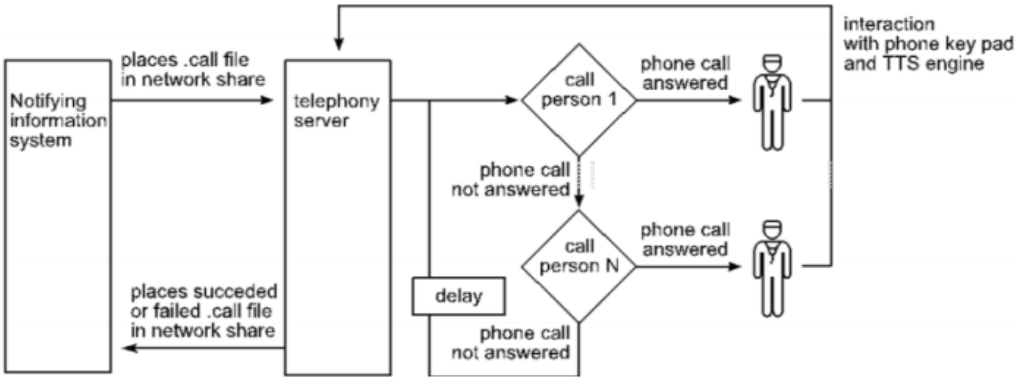

Figure 2.4: Notification process from [MSR12, Fig. 2]

Even though the channel (through which the notifications are sent) isn't necessarily the same, this solution demonstrated the importance of the confirmation and feedback of the notifications.

2.2.3 Automated Critical Test Result Notification System

Alert Notification of Critical Results (ANCR) is an application developed at the Department of Radiology, Brigham and Women's Hospital, Harvard Medical School to improve and assist in communicating critical imaging test results. In 2010, it was implemented in an institution that made use of a fully integrated EHR and a paging system, PACS, with which ANCR interacted (as shown in figure 2.5). The main features of this solution are described in table 2.1.

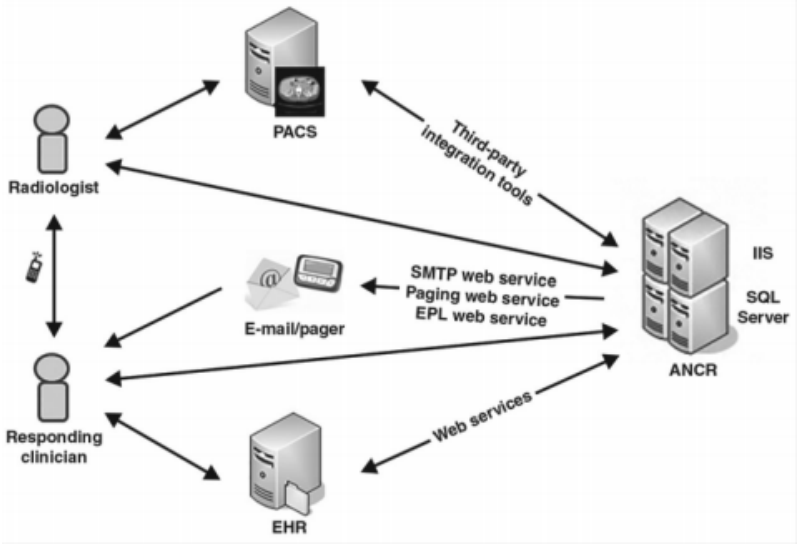

Figure 2.5: ANCR integration with clinical applications from [LOA+14, Fig. 1]

After testing, the authors sent surveys to those that used this system, both radiologists and ordering providers. Results show that while both groups believed ANCR reduces medical errors and improves quality of patient care, only radiologists felt like ANCR effectively reduced providers workload [LOA+14].

This last project is the one that is closer to solve the same issue, however, it's limited to imaging results only, excluding laboratory test and other kind of results.

2.3 Tools

To provide useful tests with relevant information related to the health-care environment, some medical semantic tools are expected to be used and consulted. Those are described in table 2.2. Those tools were chosen due to being considered the most common, well-known and used [Fra16].

At an initial stage, these tools were thought to be used to aid in the interoperability support, so that other systems could easily communicate with the rule engine, and send information in internationally accepted formats. Thus, the data models created for the rule engine database would be based on the standards specified by these tools. For example, to register a medication prescription, one would follow the format of a 'MedicationRequest' resource from FHIR Release 3 by [HL7], with property values obtained from SNOMED-CT, like 'Aspirin' as the main substance or active

Table 2.1: Key Features of ANCR extracted from [LOA+14, Table 2.1]

Features	**Description**
User authentication	Single sign-on access to multiple clinical applications after authentication at an integrated PACS workstation
Alert creation	ANCR integration with PACS enables automatic extraction of patient and examination information, reducing time and potential for manual data entry error
Alert communication	User authentication at ANCR launch enables automatic receiver paging through the hospital paging system web services
Alert acknowledgment and management	ANCR enables closing the communication loop; All unacknowledged alerts are visible on an ANCR work list
Alert reminder and escalation	ANCR sends escalation pages when urgent alerts are not acknowledged within the designated time frame according to policy; All overdue alerts are combined into a single e-mail notification to limit alert fatigue for providers who have multiple overdue alerts
Alert documentation	ANCR is part of the electronic health record, maintaining work lists for unacknowledged alerts and documentation of closed-loop communication for critical results

ingredient in the prescribed medicine. Ultimately, with the adoption of a different strategy, the only tool that was indeed used was SNOMED-CT, to create tests with valuable and health-care related information and data values.

Table 2.2: Medical Semantic tools

Type of tool	**Name of tool**
Controlled vocabularies	ICD-10-CM, the 10th edition of International Classification of Disease (ICD) by the WHO RxNorm
Taxonomy and Ontologies	The Omaha System
Reference Terminologies	Systematized Nomenclature of Medicine Clinical Terms (SNOMED-CT) Logical Observation Identifiers Names and Codes (LOINC)
Interoperability Standards	Health Level 7 (HL7)

Chapter 3

Critical Results Notifications through a Rule Engine

A Critical Result (CR) is defined as a test result presenting an important variance from normal enough to be life threatening without prompt treatment ([Lun72] as cited in [RNF+17]). Consequently, upon finding these results, the technician is responsible for reporting them back to the referred doctor, or general health professional, as soon as possible in order to begin the necessary intervention before risking the patient's safety and allowing any unnecessary damage. Since this communication depends on that technician, it isn't done through a specific channel or with a particular format or structure. Yet, besides timely, CRs must also be notified to the proper entity, and in such a clear format not to leave room for clinician misinterpretation [Sir05].

For complying with the above needs, a system to solve the issue in communicating CRs promptly and effectively is proposed. The proposed solution is based on a rule engine fed by data records which, when compared to user-created and customized rules, generate automatic notifications. These notifications are also customizable, allowing users to specify its content, to whom it should be delivered and through what channel or channels (through the webapp, mobile app and push notification, e-mail or even text message).

3.1 Requirements and Features

As mentioned, CRs may be reported through several formats and structures. Therefore, regarding their storage and process, there's two different strategies available. The first is to adhere and follow a standard (either create one, or use one that is well known and adopted, like HL7), imposing it on your users and thus, repelling those that won't be able to adapt to it. Or, alternatively, allow users to first indicate the format, sending meta-data before sending the actual data with the CRs. This second option is a feature of the proposed system. Being so, the system is generic enough to the point that it can actually be used for more than CRs. In fact, this solution can be used

for something completely unrelated to an health-care service. For example, if used properly, this solution can also be applied in a business environment where users want to be notified whenever a stock share reaches a certain value, or even in, a much simpler setting, a parking lot, notifying users whenever there are spots available.

Due to CRs urgent nature, one crucial requirement is the overall system's promptness. Apart from being fast when recording meta-data, data and rules, the system must be mainly quick on processing and analyzing rules and data, so that, afterwards, the notifications are generated readily. An upper bound limit on the time the whole analysis process might take, beginning in the new data document arrival to the last stage of receiving the notification, of five seconds seems adequate, independently of the amount of data to be analyzed.

3.2 Use cases

The several use cases this solution must provide are depicted in figure 3.1.

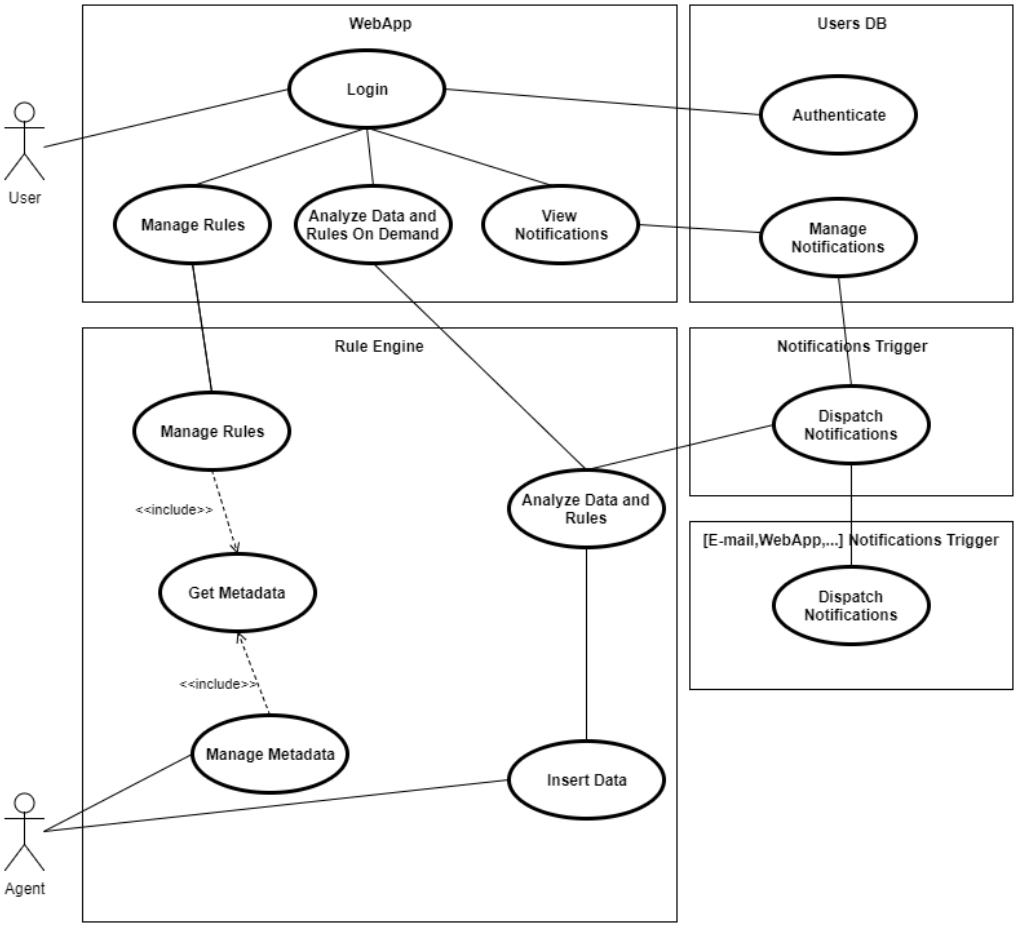

Figure 3.1: UML Use Cases diagram of the proposed solution

3.2.1 Actors

Interacting with the system, there are two very different external actors, an Agent and an User.

An Agent is responsible for providing the rule engine with the meta-data and, afterwards, feeding the engine with the data records. An User is the one who will create the rules based on the meta-data registered by the Agent, and view notifications.

3.2.2 User Stories

This subsection enumerates and describes the user stories related to the use cases, and expanding some.

1. US01

   As Agent

   Want to register meta-data

   In order to allow data and Users' rules registration according to this meta-data.

2. US02

   As Agent

   Want to register data

   In order to trigger Users' rules that use this type of data.

3. US11

   As User

   Want to authenticate into the system

   In order to get access to rules and notifications management.

4. US12

   As User

   Want to create a rule

   In order to get notifications whenever this rule is matched.

5. US13

   As User

   Want to edit a rule

6. US14

   As User

   Want to remove a rule

   In order to stop getting notifications whenever that rule is matched.

7. US15

   As User

   Want to analyze a temporary rule

   In order to test whether that rule will generate any notifications.

8. US16

   As User

   Want to analyze a data record

   In order to test whether that data will trigger any rules.

Chapter 4

RENoS Architecture

This chapter presents detailed information regarding the solution's architecture, describing its different components, how they interface with each other and how they were implemented.

Apart from the main rule engine system, the structure for a simple web application for demonstration and testing purposes was also created. Although the webapp itself never came to be developed, the underlying foundation for the app, including the database, and the methods to access, create and modify the data in it are fully working.

4.1 API-led connectivity

Application Programming Interfaces (APIs) have emerged as the most accessible way for consumers to extract value out of data [MM]. Thus it seems appropriate to use an API-led connectivity approach, i.e. a technique based on packages underlying connectivity and orchestration services as easily discoverable and reusable building blocks, exposed by APIs [Mula]. These APIs serve a specific goal – unlocking data from systems, composing data into processes, or delivering an experience [Pea].

4.2 Three-layered Architecture: System, Process and Experience layers

According to its purpose, an API can be classified into one of three layers.

APIs in the System layer aid in accessing the underlying record systems (e.g. a database), exposing the data and providing downstream insulation. Process APIs use the System ones to interact with the data independently of that data's origin and of the interaction results destination. Finally, Experience APIs, on the top layer, use the two other layers to offer personalized ways of consuming data.

The Rule Engine based Notification System (RENoS) was designed with the Three-layered architecture in mind. Figure 4.1 depicts how the different APIs fall into each layer and interact with each other.

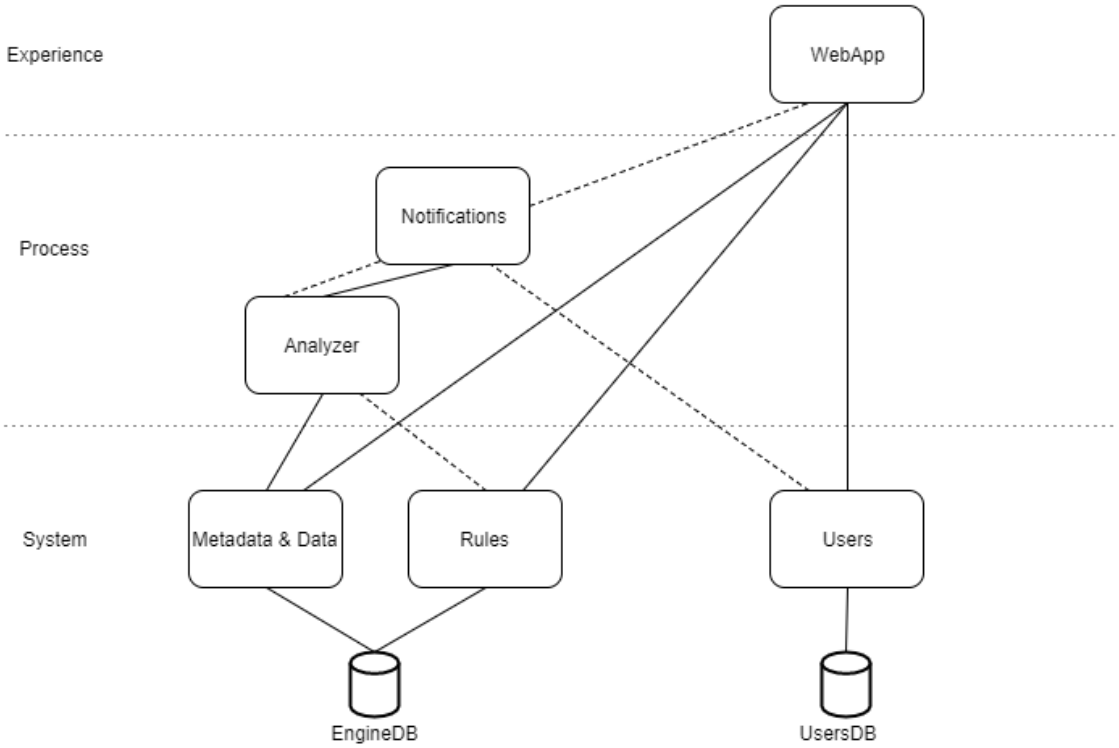

Figure 4.1: APIs layered structure

4.3 Components

Figure 4.2 depicts the main system components. These can be separated into two categories, Rule Engine components and Notification components. The following sections describe the components, mentioning its purpose.

4.3.1 Rule Engine components

Rule Engine components include components related to storing and processing meta-data, data and rules, i.e. the Agents, the Rule Engine DB, Meta-data & Data API, Rules API, Analyzer, and Unprocessed Data MQ.

The Rule Engine DB is responsible for storing the meta-data and data sent by the agents, as well as the rules created by the users. Due to this difference in the origin of these documents, the API that allows interaction with the database was separated into two APIs, Meta-data & Data and Rules. The first one provides the following features:

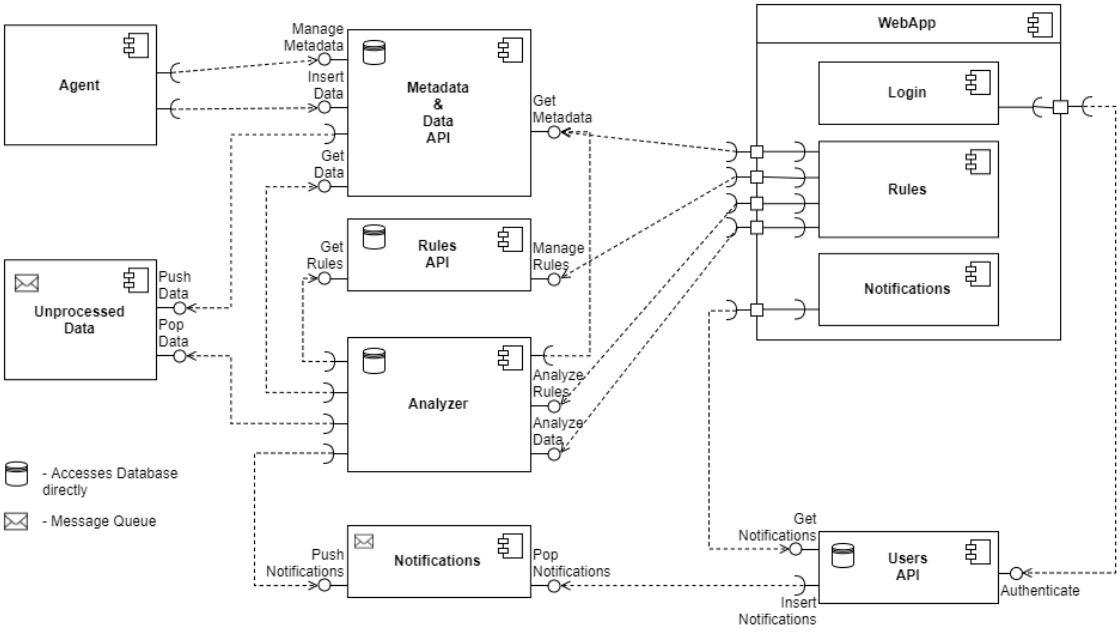

Figure 4.2: UML Component diagram with Main system components

- register and remove an agent;
- create new meta-data and, afterwards, manipulating, modifying or deleting it,
- insert new data records;
- get an individual meta-data document through its name and agent's ID, or search for several, filtering by the ID of the agent that created it, or comparing search terms with the meta-data's name, description, tags, and even properties names,
- get an individual data document through its own and agent's ID, or search for several, filtering by the ID of the agent that registered it, or comparing search terms to the corresponding meta-data's name, description, tags, and even properties names.

The latter supplies endpoints for:

- creating new rules and, afterwards, manipulating, modifying or deleting them;
- get an user's individual rule, or search for several, comparing search terms with each condition's meta-data name, or rule's description, tags,
- search for all rules that use the meta-data of a given Data.

The Analyzer, exposed through its own API, provides methods to analyze either a) a specific rule, by finding data records that match a list of conditions, or b) one data record, i.e. finding all the rules with a condition matching the data and then searching for other data records that match the rest of the conditions. Whenever an agent inserts a data record into the database, it is

redirected to the Unprocessed Data message queue (MQ). The Analyzer, being subscribed to the MQ, automatically gets the data record and processes it, like described above. After that process, the Analyzer creates and pushes the generated notifications to another queue, the Notifications MQ.

4.3.2 Notification components

Notification components consist of the Users DB, Authentication and Notifications APIs, Notifications MQ and WebApp. These components were solely created for demonstration and testing purposes. Therefore, it's implementation is mostly simple and basic.

The Users DB stores basic User login information, i.e. the user-name and password, and each user's notifications. Interaction with the DB is done through its own API, providing methods to authenticate an user and insert and get notifications.

As previously mentioned, although the WebApp was never in fact developed, the system is completely ready for it, since its fundamental base and foundation, consisting of all the previous components and APIs, is fully functioning and was built with the app interaction in mind. The app is included among the other components, and in the diagrams as well, to demonstrate how it should make use of and exchange information with the other components.

4.3.3 Component interaction

Some of the interactions between the different components, described in the previous subsections, are exhibited in the sequence diagrams in the figures 4.3, 4.4, 4.5 and 4.6.

4.4 Physical Systems

Even though, during testing, all components were run in one machine (with Windows 10, 4GB of memory and and i3@2.40GHz processor), RENoS was made with modularity and component independence in mind. Therefore, the components could be distributed among several machines, greatly increasing overall system's performance. One possible arrangement would be:

- one machine per created Agent,
- another one dedicated to the Rule Engine DB, the Meta-data & Data and Rules APIs,
- another for the Unprocessed Data MQ,
- one or more machines, depending on workload, running the Analyzer,
- another for the Notifications MQ, and finally,
- one as a server for the Users DB, its API and the WebApp.

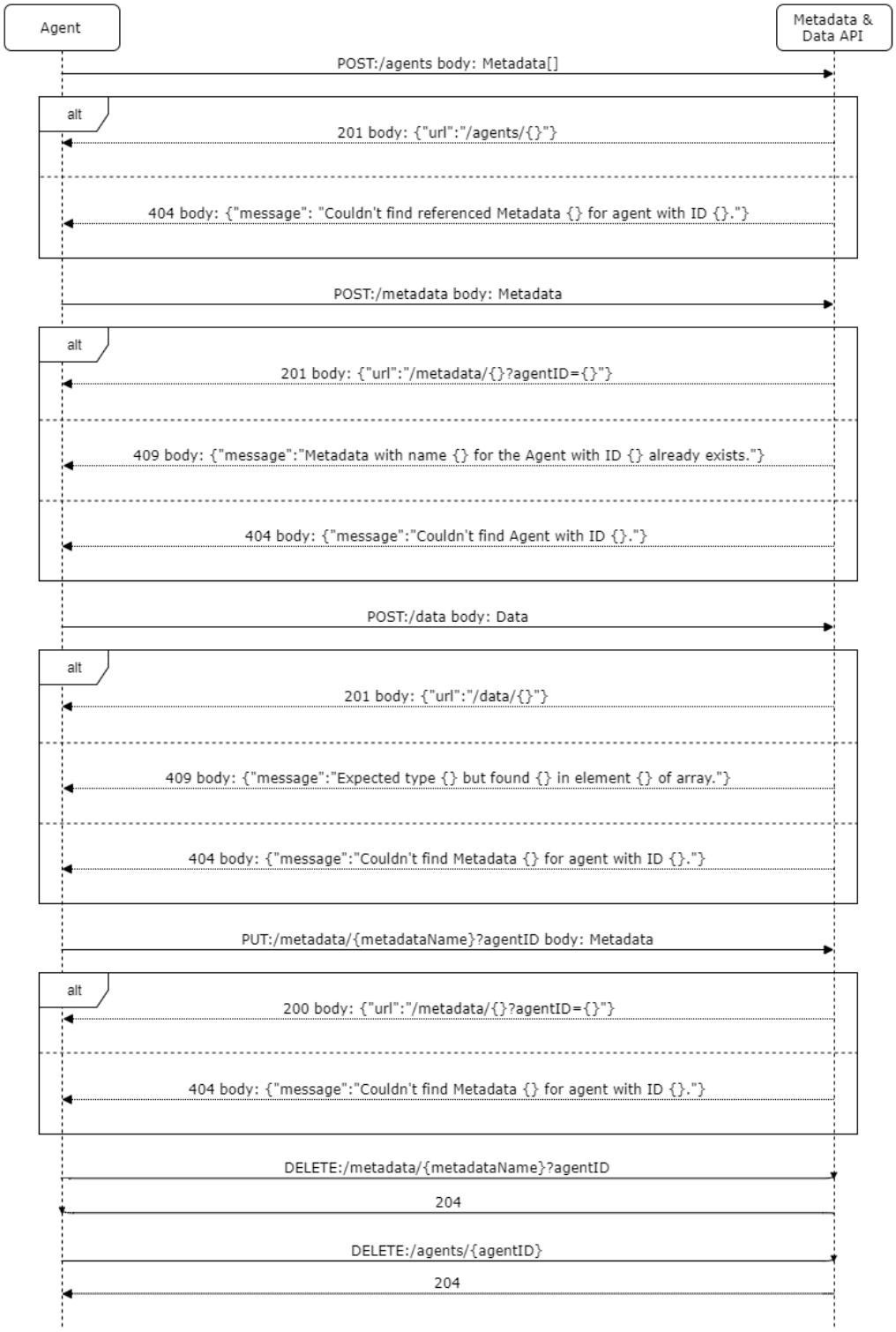

Figure 4.3: UML sequence diagram of Agent and Meta-data & Data API interaction

Figure 4.4: UML sequence diagram of App and Metadata & Data API interaction

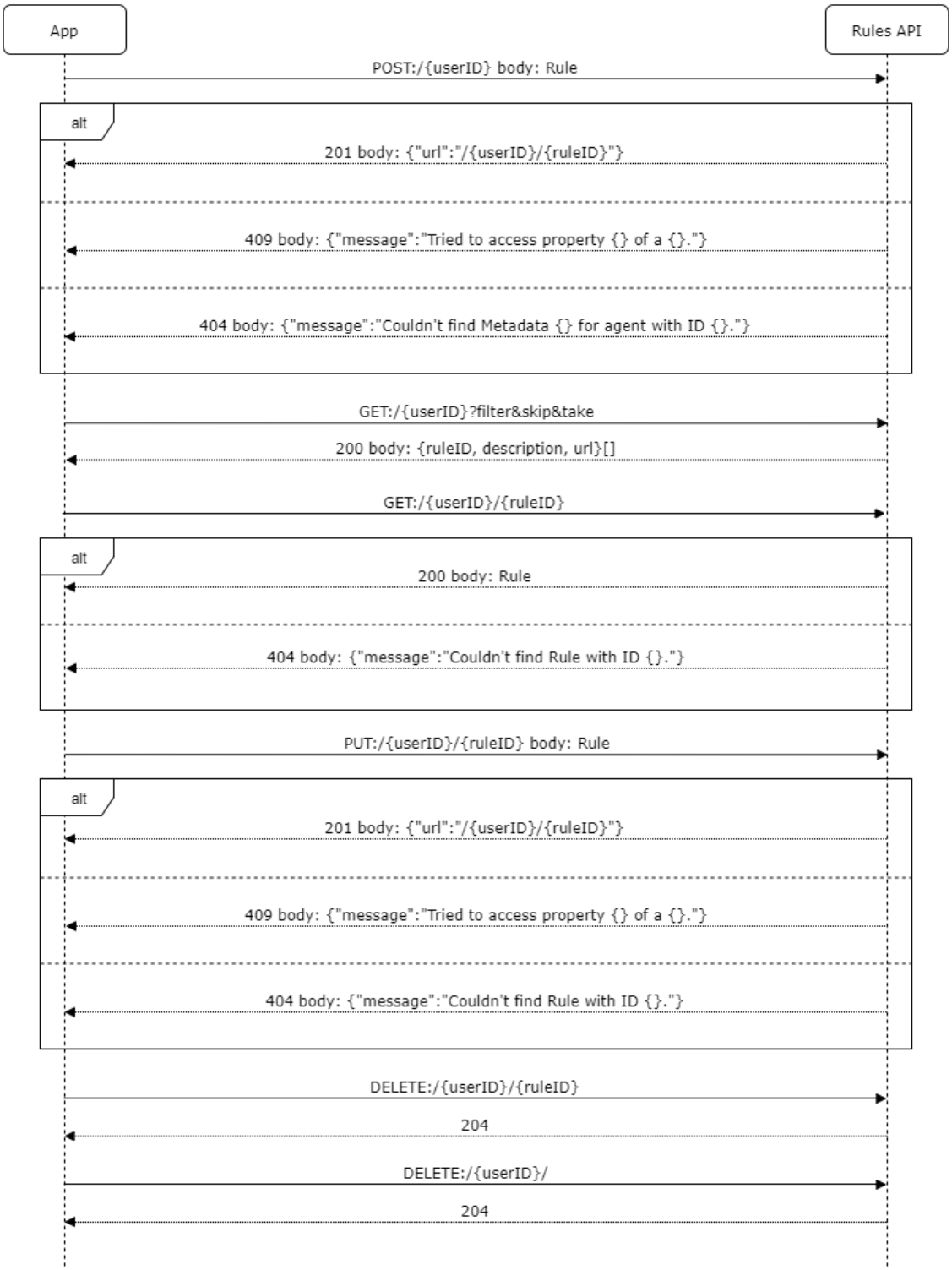

Figure 4.5: UML sequence diagram of App and Metadata & Data API interaction

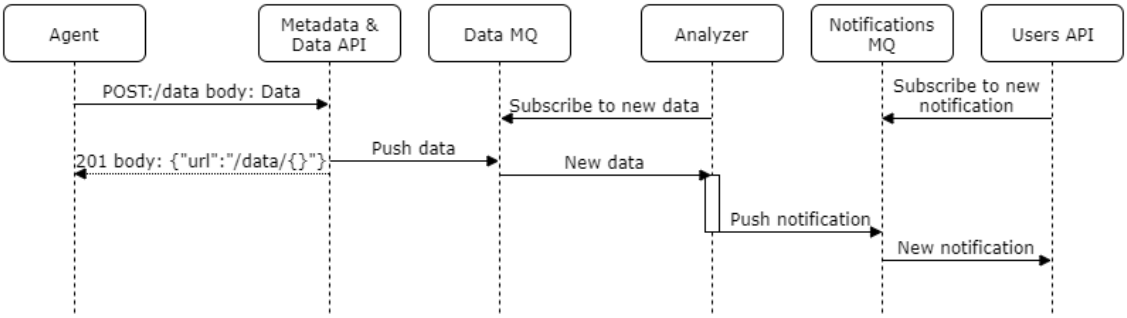

Figure 4.6: UML sequence diagram of overall system components communication

Chapter 5

RENoS Implementation

This chapter provides descriptions for the main tools, frameworks or technologies, and algorithms, methodologies or strategies used during the development and implementation of the solution, as well as characterizations of the data models designed for the two databases.

5.1 Used Technologies and Tools

One of the most important stages in building an application happens before actually implementing any code, it's researching and choosing the different technologies and tools the application will make use of and depend on. When looking for any tool, there are some general requirements it should meet. It should be preferentially open-source, have good support through an active community as well as a thorough and clear documentation, including starting guides, tutorials, reference and API documentation.

5.1.1 Database Management System

The first choice was the Database Management System (DBMS). Apart from the general requirements, when searching for a DBMS, one should look for one that also has good driver support. Due to the flexibility of the documents' structure and the need for faster operations, this application would benefit from the use of a non-relational database. Three extremely popular solutions that meet all these requirements were considered: Apache Cassandra [Cas], Redis [Red], and MongoDB [Mona]. These kind of database solutions can be classified by their data-model, falling into one of four categories: Document, Graph, Key-value and Column models. Coming from an object-oriented programming background, the Document model is the most familiar and easier to work with. MongoDB being the only one that follows it (the others using Column and Key-value,

respectively), was the chosen DBMS.[1] To aid in the database interaction and management, the MongoDB Compass graphical application was used.

Regarding the secondary database, the one dedicated to storing user and notification information, due to having a much simpler data model and no need for schema flexibility, such an atypical solution wasn't required. Therefore, available options included more traditional tools such as MySQL and Microsoft SQL Server. Since there's no crucial difference between those, the author opted for the one with most past experience, MySQL.

5.1.2 Enterprise Service Bus

In order to effortlessly integrate and establish the communication between all components, an Enterprise Service Bus (ESB) was recommended. Two great solutions are Mule ESB [Mulb] and Apache Camel [Cam]. Both have an active community, good support and, more importantly, well documented, easily configurable and feature-rich MongoDB connector/component, which highly simplifies the database integration. A major advantage of Mule ESB is in the IDE. Whereas Apache offers no dedicated IDE for Camel (all configuration is done through any regular text editor), Mulesoft, the company behind Mule ESB, provides Anypoint Studio, a graphical IDE with a drag-and-drop environment, making it easy and intuitive to integrate several applications. For this reason, the Mule ESB and associated tools were used.

5.1.3 Advanced Message Queuing Protocol implementation

To increase performance, some of the data processing can be done in a asynchronous and distributed way. One way to accomplish that is using an Advanced Message Queuing Protocol (AMQP) implementation in a Competing Consumers pattern. Two AMQP brokers are RabbitMQ [Rab] and Apache ActiveMQ [Act]. The factor which influenced the choice between them is the AMQP supported version. While RabbitMQ supports AMQP v0.9, ActiveMQ only has AMQP v1.0 support. Even though both versions were sufficient to achieve my goal, the chosen ESB, Mule ESB, already provides an AMQP V0.9 connector. Therefore RabbitMQ was the more suitable solution.

5.2 Docker images

Some of the tools, more specifically MySQL and RabbitMQ, were installed through Docker images, and are ran in containers. Docker containers are running instances of images, which are defined as stand-alone, executable packages that include everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and configuration files [Doc],

[1] Other requirements such as replication and sharding or partitioning were not taken into consideration due to not being applicable.

heavily assisting in the installation and configuration phases. These are comparable to virtual machines, but have better performance. Since these images are isolated from each other and the host, in order to communicate between them, the ports each use had to be exposed to the host machine.

5.3 Data models

For this solution, two data models were designed, the one for the Rule Engine DB (depicted in figure 5.1) and the Users DB (depicted in figure 5.2).

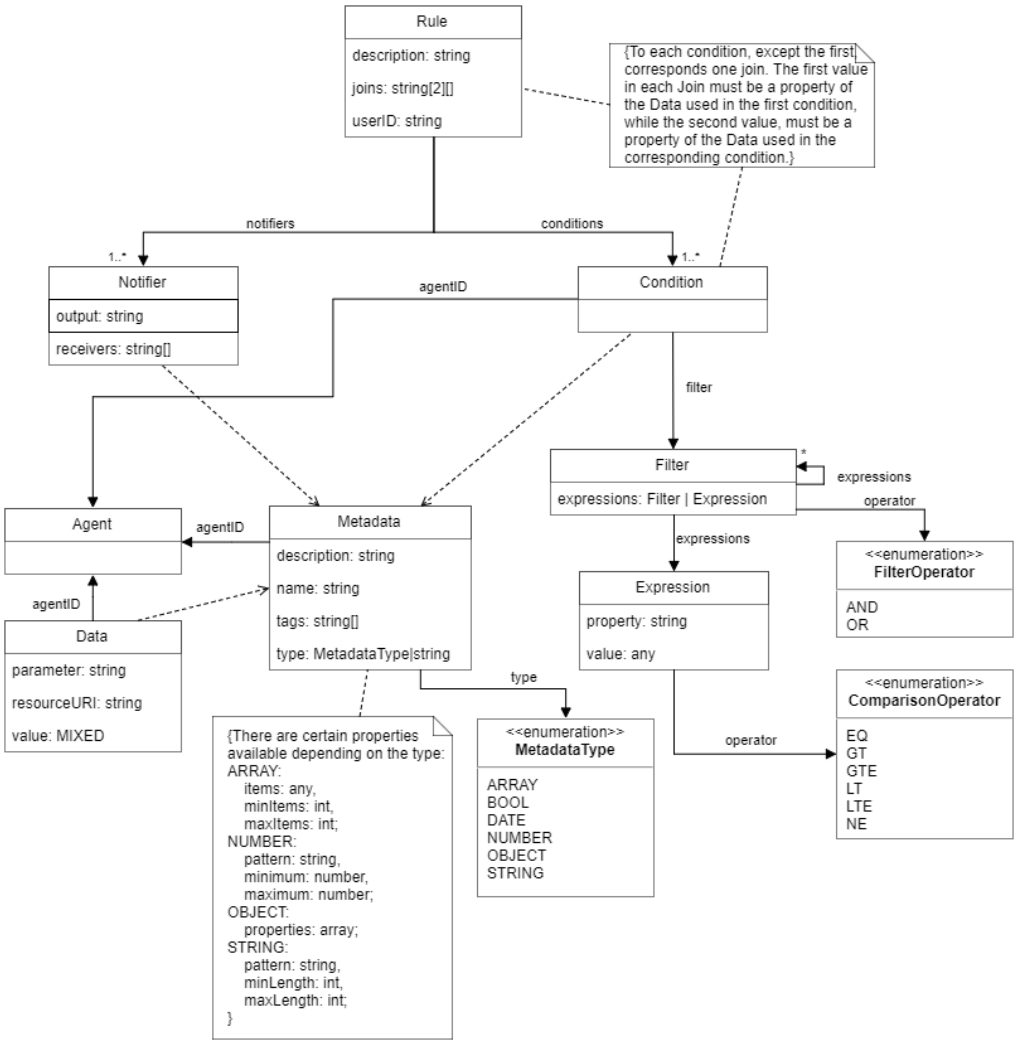

Figure 5.1: UML Class diagram of Rule Engine DB

The first data model represents models for data, meta-data and rules.

The meta-data document dictates the structure or format of corresponding data documents. Through the 'type' field , meta-data can represent data of basic types ('bool', 'date', 'number' or 'string'), complex types that contain other types ('array' or 'object') or even another meta-data by referencing it. Depending on the type, some properties become available that impose some

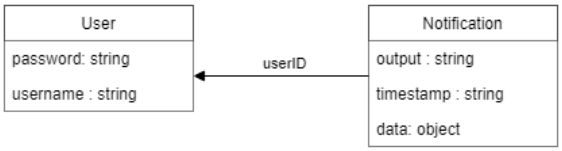

Figure 5.2: UML Class diagram of Users DB

restrictions on the represented data, for example, for an array, one can specify the type of the items, through the 'items' field, or even the minimum and maximum number of items, through the 'minItems' and 'maxItems' fields, respectively. There are two examples of meta-data in listing 5.1.

All data documents of the same meta-data must have a 'value' field that follows that meta-data format, e.g. if a meta-data document is specified with the 'bool' type, the data document of this meta-data must have a 'value' property of the boolean type. Listing 5.2 provides two examples of data for the previous meta-data.

Rule documents contain conditions, joins and a reference to the user who created it.

A condition includes a reference to a meta-data and the matching filter to be applied to the data documents of that meta-data. This filter is made of a logical operator ('AND' or 'OR') and one or several other filters or expressions to be evaluated together. These expressions specify a property of the condition's meta-data and a value to be compared together through a relational operator ('EQ' for equality and 'GT', 'GTE', 'LT', 'LTE', 'NE' for inequalities).

A join contains two references to properties of the data 'value' field in the corresponding conditions. All conditions are joined to the first one, i.e. the first join links the first and second conditions, second join links first and third conditions, and so forth. These properties' values must be equal in both data in order to join them.

For a better understanding, a rule's conditions and joins can be interpreted in a structure similar to a graph, in which each condition, represented as a node, has an inner graph itself representing its filter, and his connected to other conditions through the join properties. This representation is seen in figure 5.3 for the rule shown in the listing 5.3 with the addition of another condition for Urinalysis lab tests.

The second, simpler data model, includes models for the users (consisting of a user-name and a password) and its notifications (with the message content, time-stamp of creation and the data documents that triggered the notification attached).

5.4 Rule analysis

Analyzing a rule consists of searching for data records that, when joined together, match the conditions, filters and expressions inside the rule. In order to do that, the Analyzer executes a MongoDB aggregation [Monb]. MongoDB aggregations are modeled after processing pipelines. Documents are passed along the multiple stages that transform them into an aggregated result.

```
[
  {
    "agentID": "5a3935820c19c6068c3c9626",
    "name": "person",
    "type": "object",
    "properties": [
      {
        "name": "id",
        "type": "number"
      },
      {
        "name": "name",
        "type": "string"
      },
      {
        "name": "age",
        "type": "number",
        "units": "years"
      },
      {
        "name": "gender",
        "type": "string",
        "enum": [
          "FEMALE",
          "MALE"
        ]
      }
    ]
  },
  {
    "agentID": "5a5fe5298a2b36edb431f6e1",
    "name": "prescription",
    "type": "object",
    "properties": [
      {
        "name": "patient",
        "type": "number"
      },
      {
        "name": "medication",
        "type": "string"
      }
    ]
  }
]
```

Listing 5.1: Example of two Meta-data documents that represent a person and a medication prescription, respectively.

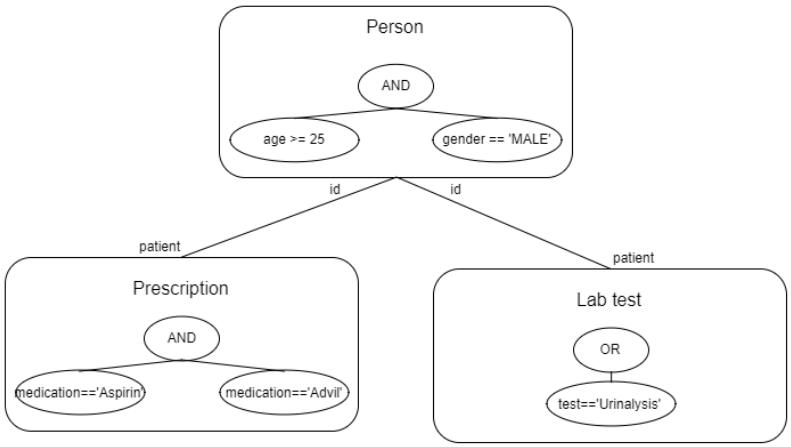

Figure 5.3: Representation of a rule in a graph

The main stages of the created aggregation are '$match' and '$lookup'. The former one is used to filter and relay to the next stage only those documents that match specific conditions, whereas the latter runs a join operation, adding an array with matched documents into each input document.

For each condition, a match stage is generated, which filters the referenced meta-data and any property conditions. Listing 5.4 offers an example of the match document for the first condition in listing 5.3. When a rule joins several conditions, a lookup stage is added for each one. This join operation was specially easier with the use of the recent lookup syntax, which allows defining variables and an inner pipeline (that can use those variables) to run on the joined documents. Inside this inner pipeline are the following stages:

1. a match to compare the properties specified in the join;
2. another match to apply the filter in the condition;
3. a sort by time-stamp, ordering from most recent to oldest,
4. a limit by 1 document.

The two latest stages, were added in an attempt to improve performance, by including only the newest data document that matches the condition, instead of all of them.

Listings 5.4 and 5.5 exhibit the generated match and lookup stages for the conditions and join in listing 5.3, respectively. Finally, listing 5.6 shows what the result of the aggregation generated from the rule would be with the previous data examples.

5.5 Data analysis

In order to increase performance, instead of querying each rule constantly, the system analyzes each of the incoming data documents, following the process described next:

1. search for rules that have conditions on meta-data of the received data,

```
[
  {
    "_id": { ObjectId("5a6001c2d5eccc214c006c78") },
    "agentID": "5a3935820c19c6068c3c9626",
    "metadata": "person",
    "value": {
      "id": 1,
      "name": "Arthur Ace",
      "age": 50,
      "gender": "MALE"
    },
    "timestamp": "2010-01-01T00:00:00"
  },
  {
    "_id": { ObjectId("5a6001c2d5eccc214c006c7a") },
    "agentID": "5a5fe5298a2b36edb431f6e1",
    "metadata": "prescription",
    "value": {
      "patient": 1,
      "medication": "Aspirin"
    },
    "timestamp": "2017-12-20T11:12:56"
  }
]
```

Listing 5.2: Example of two Data documents for the previous Meta-data.

2. from those rules, filter only those whose conditions are met by the received data,

3. run a modified aggregation:

   - if the matched condition is the first one:
     (a) the first match filter becomes a filter by '_id' of the Data document,
     (b) lookups' match filters use values from the received Data
   - else
     (a) the first match filter uses the value of the join property in the received Data,
     (b) the match filter corresponding to the condition becomes a filter by '_id' of the Data document

This process allows to filter beforehand many unnecessary Data documents. The different generated aggregations are illustrated in listings 5.7 and 5.8 for the two kinds of Data analysis. This Data analysis, through either of the mentioned aggregations (based on either the 'person' or 'prescription' Data documents) produces the same result as the one on listing 5.6.

```
{
  "conditions": [
    {
      "agentID": "5a3935820c19c6068c3c9626",
      "metadata": "person",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "GTE",
            "property": "age",
            "value": 25
          },
          {
            "operator": "EQ",
            "property": "gender",
            "value": "MALE"
          }
        ]
      }
    },
    {
      "agentID": "5a5fe5298a2b36edb431f6e1",
      "metadata": "prescription",
      "filter": {
        "operator": "OR",
        "expressions": [
          {
            "operator": "EQ",
            "property": "medication",
            "value": "Aspirin"
          },
          {
            "operator": "EQ",
            "property": "medication",
            "value": "Advil"
          }
        ]
      }
    }
  ],
  "joins": [
    [
      "id",
      "patient"
    ]
  ]
}
```

Listing 5.3: Example of a rule document, one for at least 25 years old males who have been prescribed Aspirin or Advil.

```
{
  "$match": {
    "agentID": "5a3935810c19c6068c3c9624",
    "metadata": "person",
    "$and": [
      {
        "value.age": {
          "$gte": 25.0
        }
      },
      {
        "value.gender": {
          "$eq": "MALE"
        }
      }
    ]
  }
}
```

Listing 5.4: Corresponding $match stage of the first condition in listing 5.3 when executing a Rule Analysis.

5.6 Database indexes

As previously mentioned, a primary concern of this system is the time it takes to process and analyze each data record. As a consequence, the use of indexes would aid immensely meeting this requirement. MongoDB indexes are the reason behind its fast queries. If an adequate index isn't used, MongoDB must scan the whole list of documents in a collection to find those that comply with the query. On the other hand, using the right index, might reduce significantly the number of examined documents [Monc].

MongoDB also allows the creation of partial indexes [Mond]. These partial indexes enable indexing only documents of a collection that match certain conditions, i.e. meet a regular filter expression, resulting in less used memory space and better performance upon index creation and throughout its maintenance.

Particularly these kind of indexes would be extremely helpful. Since all Data is in one collection, instead of being distributed in several, it would allow to index only documents of certain Agent and Meta-data name. Unfortunately, MongoDB still has some limitations regarding partial indexes. It won't allow the creation of an index with the same definition (indexing same document properties) as an already existing index, even if it has a different partial (indexing different documents than the ones already indexed). A useful illustration of when these indexes would be used is adding a meta-data that represents performed lab tests, which includes a 'patient' property (equal to the 'patient' property in the 'prescription' meta-data in listing 5.1). Due to this issue, partial indexes could not be used and regular indexes were resorted to, instead.

These indexes are created upon inserting a new or updating an old rule and are based on the properties used in each condition of the specified rule (in ascending order), e.g. for the first

```
{
  "$lookup": {
    "from": "data",
    "let": {
      "patient": "$value.id"
    },
    "pipeline": [
      {
        "$match": {
          "$expr": {
            "$eq": [
              "$value.patient",
              "$$patient"
            ]
          }
        }
      },
      {
        "$match": {
          "agentID": "5a3935810c19c6068c3c9624",
          "metadata": "prescription",
          "$or": [
            {
              "value.medication": {
                "$eq": "Aspirin"
              }
            },
            {
              "value.medication": {
                "$eq": "Advil"
              }
            }
          ]
        }
      }
    ]
  }
}
```

Listing 5.5: Corresponding $lookup stage of the second condition and respective join in listing 5.3 when executing a Rule Analysis.

```
{
  "_id": { ObjectId("5a6001c2d5eccc214c006c78") },
  "agentID": "5a3935810c19c6068c3c9624",
  "metadata": "person",
  "value": {
    "id": 1,
    "name": "Arthur Ace",
    "age": 50,
    "gender": "MALE"
  },
  "timestamp": "2010-01-01T00:00:00",
  "prescription": [
    {
      "_id": { ObjectId("5a6001c2d5eccc214c006c7a") },
      "agentID": "5a3935810c19c6068c3c9624",
      "metadata": "prescription",
      "value": {
        "patient": 1,
        "medication": "Aspirin"
      },
      "timestamp": "2017-12-20T11:12:56"
    }
  ]
}
```

Listing 5.6: Result of the Rule Analysis of the Rule in 5.3 with the Data in 5.2 in the database.

condition in listing 5.3, would be generated an index that sorts data based on its 'value.age' and 'value.gender' properties.

5.7 Utilities library

There are some specific functionalities or parts of these components that are either not natively supported, or involve logic too complex to be implemented in Mule ESB. For those cases, a Java library was created to be shared among the remaining projects, which included classes providing methods that are called through Mule Invoke components. More specifically, these functionalities include:

- rule analysis (building the several aggregation stages and, afterwards, running it through the Java driver for MongoDB);
- metadata, data and rule validations, i.e. assessing that data and rules are well built and follow the used meta-data patterns; and checking whether a meta-data refers a non-existing type,
- indexes and notification content generation.

```
[
  {
    "$match": {
      "_id": { ObjectId("5a6001c2d5eccc214c006c78") }
    }
  },
  {
    "$lookup": {
      "from": "data",
      "pipeline": [
        {
          "$match": {
            "agentID": "5a3935810c19c6068c3c9624",
            "metadata": "prescription",
            "$or": [
              {
                "value.medication": {
                  "$eq": "Aspirin"
                }
              },
              {
                "value.medication": {
                  "$eq": "Advil"
                }
              }
            ],
            "value.patient": 1
          }
        }
      ]
    }
  }
]
```

Listing 5.7: Generated aggregation from the Data Analysis of the 'person' document in 5.2 for the Rule in listing 5.3.

```
[
  {
    "$match": {
      "agentID": "5a3935810c19c6068c3c9624",
      "metadata": "person",
      "$and": [
        {
          "value.age": {
            "$gte": 25.0
          }
        },
        {
          "value.gender": {
            "$eq": "MALE"
          }
        }
      ],
      "value.id": 1
    }
  },
  {
    "$lookup": {
      "from": "data",
      "pipeline": [
        {
          "$match": {
            "_id": { ObjectId("5a6001c2d5eccc214c006c7a") }
          }
        }
      ]
    }
  }
]
```

Listing 5.8: Generated aggregation from the Data Analysis of the 'prescription' document in 5.2 for the Rule in listing 5.3.

Chapter 6

Tests and Results

There are some interesting aspects to be extracted from testing. The main intention behind it is to see how the system scales, meaning how it performs when the quantity of information keeps rising drastically. But one can also see where the system is spending more time, in order to try and identify flawed parts, processes, algorithms of the solution and search for alternatives. Other purposes are to compare different adopted strategies regarding a specific issue or simply see how one implementation detail has an impact on the overall performance.

6.1 Tests operation and execution

For this solution in particular, it's of interest to see:

1. how rule complexity affects performance,
2. how the addition of the two sort and limit stages to the lookup pipeline affected performance;
3. how effective are the rule-oriented generated indexes;
4. the difference between durations of rule analysis and data analysis;
5. if the use of Mule ESB has much impact on the Mongo aggregation;
6. how the number of registered data records affects the system;
7. how is the time taken distributed among the different 'processes' from the arrival of the data to the notification insertion as well as the total duration.

In order to extrapolate that information, Mule Logger components, that also output the current time to the millisecond, were added in the following events:

1. upon arrival of a new data record (through an HTTP POST request) to be inserted in the DB;

2. after the data is inserted and sent to the MQ to be analyzed;
3. upon receiving a data record (through the MQ) to be analyzed;
4. before the search for rules that use the data;
5. when the results of the search are retrieved;
6. before analyzing a rule;
7. after a rule analysis ends;
8. after sending the notification to the MQ to be inserted;
9. upon receiving a notification through the MQ to be inserted in the DB;
10. after a notification is finally inserted in the DB;

These logs were inserted among other default Mule ESB logs, thus two Java applications (one for rule analysis and another for data analysis) were made to filter and parse these logs. While the first only outputs a .csv file with the rule analysis duration, the second application, also includes the durations, in milliseconds, of the following phases:

1. inserting the data into the DB (events 2-1);
2. data in the MQ, or Meta-data to Analyzer APIs latency (events 3-2);
3. finding rules (events 5-4);
4. analyzing rule (events 7-6);
5. total Analyzer duration (events 8-3);
6. notification in the MQ, or Analyzer to Users APIs latency (events 9-8);
7. inserting the notification into the DB (events 10-9);
8. total time (events 10-1);

To test these aspects, the database was first populated with 1 million data documents. This database population was done through another Mule project specifically created to simulate three different agents, one for person, another for medication prescription and the last one for lab test requisition documents. This application provides four endpoints to generate either:

1. 50 semi-random people, whose names are taken from a static list, genders are according to the name and age varies randomly between 10 and 90 years, including an auto-incrementing id;
2. a prescription with:

- a random person selected as the patient, through the id;
- a random medicine from a list of six,
- a random dosage form from a list of four;
- a random prescription date of up to a year from the current date;
- an expiration date by adding 30 days to the prescription date;

3. a random lab test with a random person selected as the patient (through the id) and a random test name from a list of seven;

After that, six rules were created with different complexity. The first three of them are made of two conditions and one join (linking a person to either a prescription or a test), while the rest have three conditions and two joins (linking a person to a prescription and a lab test). For a better understanding of the complexity of the rules, table 6.1 presents descriptions for each rule, and appendix A includes the documents used to build them. Creating these rules, automatically created the respective indexes.

Table 6.1: Description of each rule analyzed

Rule	Description
1	80 or more years old males who have been prescribed a medication with the Aspirin medicine
2	people of the female gender who have been requested a Lipid panel test
3	people between the age of 20 and 50 years old who have been prescribed medication of Aspirin or Ibuprofen medicines
4	males that have been prescribed medications of Desloratadine under the Aerosol form, and requested a Urinalysis
5	females of 45 or more years old that have been prescribed medications under the Syrup or Pills forms, and requested a Urinalysis
6	females younger than 40 years old who have been prescribed a Etofenamate medication, and request a Complete blood count test, both after December 21st, 2017

Having the database ready, all rules were analyzed sequentially (rules 1 through 6) five times in order to obtain an average of the analysis duration per rule, trying to eliminate other factors that might interfere with the results. After that, the process was repeated for data analysis, by inserting data documents that would trigger each rule.

Following both analyses for the two variants of the two initial tests (with and without the stages, with and without the indexes), the built aggregations for each rule were ran in the MongoDB shell. Next, data documents, of the prescription and lab test types, were removed so that only 500 thousand were present in the database. Both analyses were run as described before (but only with the stages and indexes present) and the process was repeated for 100 thousand, 50 thousand, 10 thousand and five thousand documents.

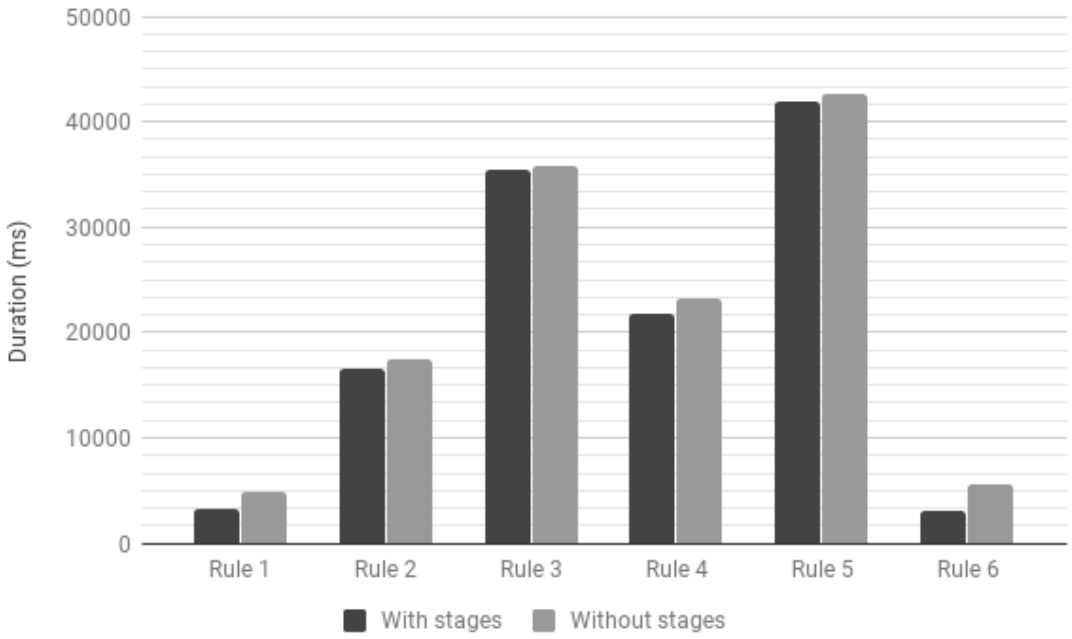

Figure 6.1: Rule analysis duration with and without $sort and $limit stages comparison (in ms)

6.2 Execution Results

In this section are depicted the graphs representing the obtained data from the mentioned tests. The tables in the appendix B present the absolute values that were used to create these graphs.

The first test ran was to determine how the stages influence the performance. In order not to tamper with the results, this test was run with the indexes enabled in both cases. The results can be seen in figures 6.1 and 6.2 for rule and data analyses, respectively.

After that, the next goal was to see how much the indexes created per condition were reducing the analyses duration times. For this test, all analyses were done with the sort and limit stages in the aggregations. The graphs in figures 6.3 and 6.4 show the obtained results for rule and data analyses, respectively.

The following aspect to be determined was the difference between the durations of rule and data analyses. This comparison is done through the times obtained while running both analyses with the two stages and auto-generated indexes from 6.1 and 6.2. The contrast between them is show in figure 6.5.

The comparison of the MongoDB aggregations using the Mule ESB (through the Java driver) and the Mongo shell, in both cases done with both the indexes and the stages, can be seen in the graph portrayed in figure 6.6.

The scaling ability of this application was evaluated for both the rule and data analyses, which can be seen in figures 6.7 and 6.8. To have constant tests and resulting values, all analyses for the

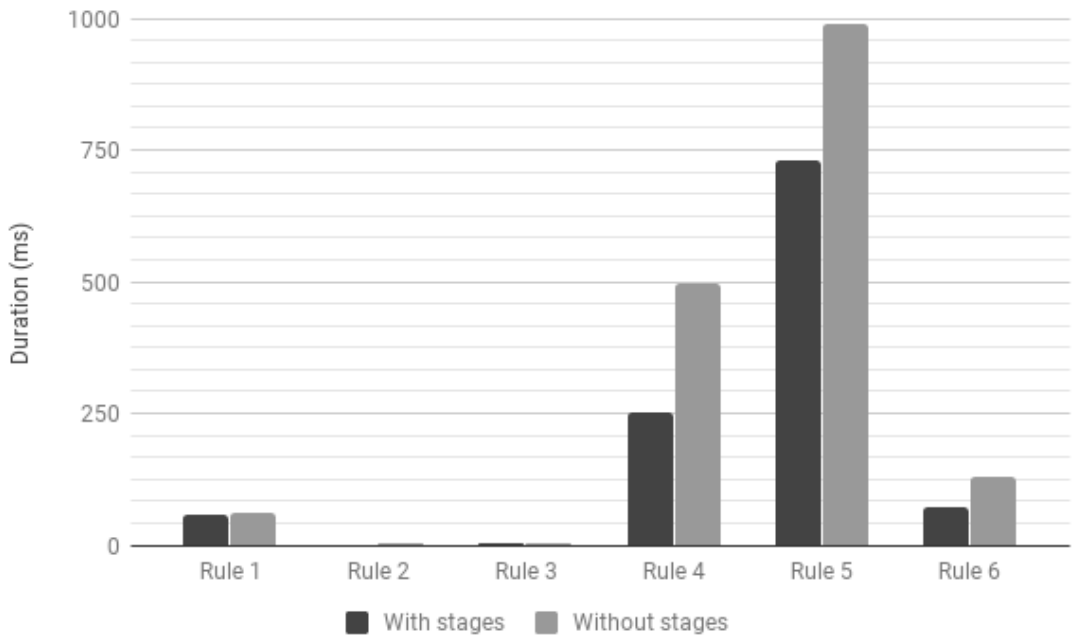

Figure 6.2: Data analysis duration with and without $sort and $limit stages comparison (in ms)

scaling test were executed with the indexes enabled in the DB, and the mentioned stages included in the generated aggregations.

The distribution of the different data analysis phases durations, since the reception of the data record to the insertion of the notifications in the database, is represented in the figure 6.9. The analysis itself, from the data arrival to the Analyzer (from the MQ) up until the creation of the notifications with the found matched data, can be better separated. Its distribution is seen in figure 6.10.

6.3 Discussion of Results

Despite, as previously mentioned, running the analyses five times to attenuate any possible external influence, there are some that will always have an impact. For example, rule 1 constantly has much higher values when compared to both the other rules with two conditions, as show in 6.8. The fact that it is always the first to be analyzed, probably awakening initiation processes in Mule ESB as well as MongoDB, might have contributed to this difference. To further mitigate this problem, without influencing the results the other way around, meaning showing false faster results, one could double (or even more) the times the analyses were run, or change the initial rule in between tests.

Also, for a more real use case scenario, data documents should be of more types than just persons, medication prescriptions and lab tests requisitions.

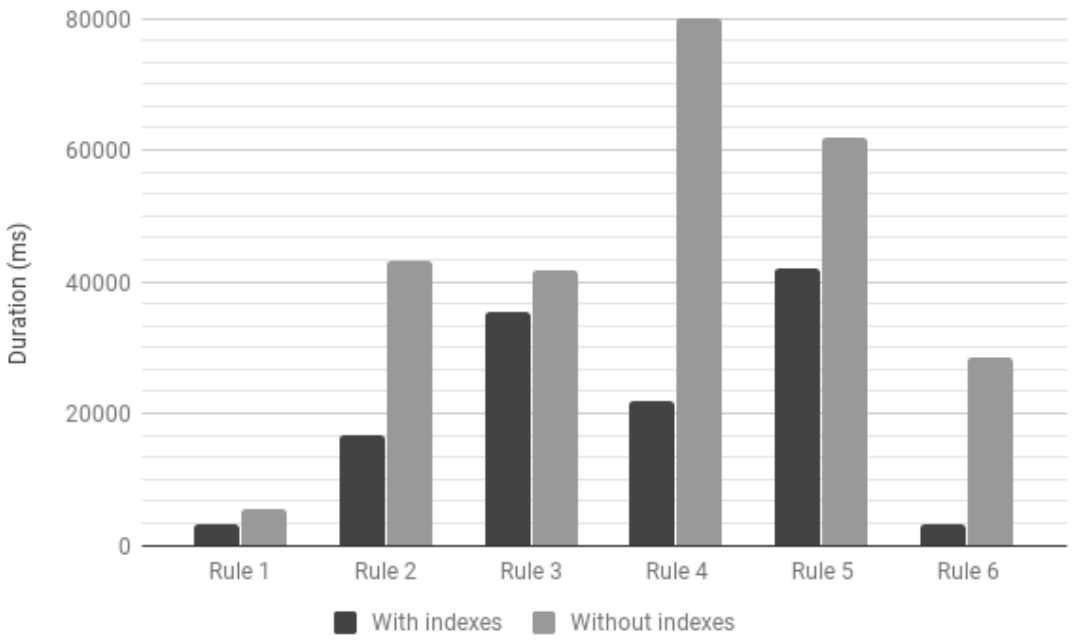

Figure 6.3: Rule analysis duration with and without indexes comparison (in ms)

In the following subsections, comments and discussions regarding the different evaluated aspects are presented.

6.3.1 Rule complexity

By analyzing the graphs, it seems that rule complexity is correlated, at least to some level, to analyses duration. Increasing complexity may lead to higher times. The sixth rule seems to be the exception. The fact that rule 6 is the only one that has conditions that compare string values with the 'greater than' operator is one of the possible reasons on why its duration is so different from the other rules with three conditions.

6.3.2 Sort and Limit stages

The graphs in figures 6.1 and 6.2 show that the addition of the sort and limit stages inside the lookup's pipeline, so that lookups return only the most recent data document, reduced analysis times, more so on the data than the rule analysis. This improvement seems illogical at first, since its adding two other stages to the aggregation per lookup, and consequently adding more workload to the database, but the amount of documents passed around the stages has an impact on available memory, specially when running aggregations with a million documents in the database. Also due to this particular reason, its reasonable that these stages have more impact on rules with a higher number of joins, as seen in the rules 4 and 5 of the data analysis. Even though it mostly improves analyses times, it actually hinderer the data analysis for rule 3. Overall, adding these

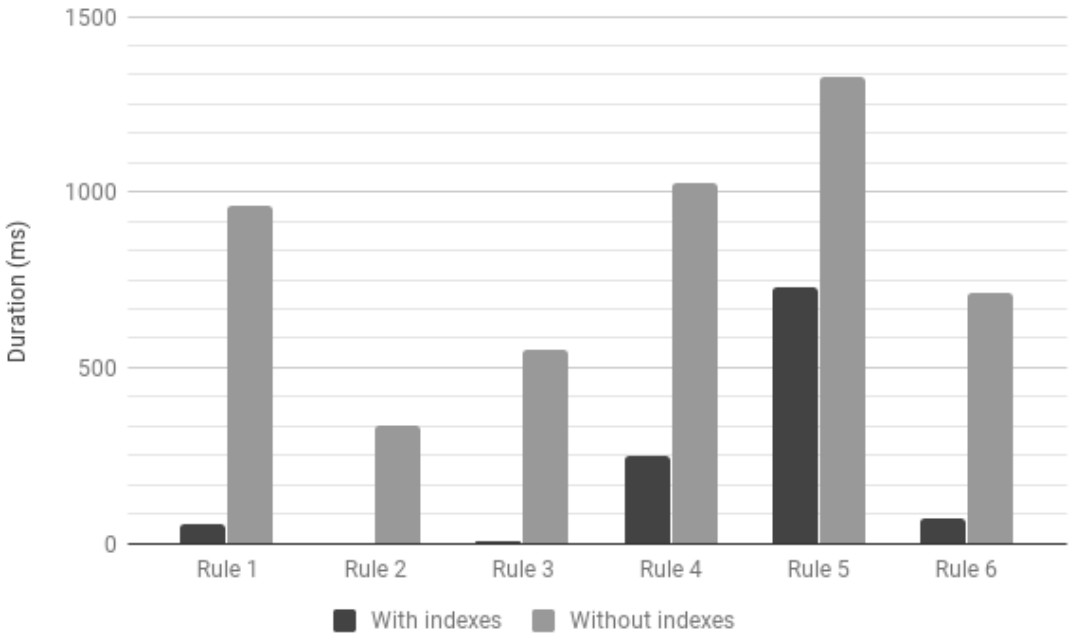

Figure 6.4: Data analysis duration with and without indexes comparison (in ms)

stages, reduced rule analysis times from 1% to 43%, and reduced data analysis' 49% but also increased it, in the case of rule 3, by 12%.

6.3.3 Indexes

As mentioned in the previous chapter, MongoDB indexes have a huge impact on the quickness of its queries. Consequently, it's expected that adding indexes to the collection significantly lowers the analyses times, for both rule and data. Adding the indexes improved rule analysis duration from 13% up to 99%, with an average of 84%, and data analysis from 45% up to 89%, with a 52% average. This impact is depicted in the figures 6.3 and 6.4. Logically, the benefits of adding these indexes are more visible in analyses that take more time, therefore the difference between the average improvements for rule and data analyses is to be anticipated.

6.3.4 Rule and data analyses comparison

Data analysis is extraordinarily faster than rule analysis. As shown by the graph in figure 6.5, data analysis takes an average of 99% less time than the corresponding rule analysis. However, this enhancement can be influenced by some factors that weren't tested, for example, if the data being analyzed corresponds to the first or the remaining conditions. In the schema tested, it didn't make sense to register a new person record, because it wouldn't trigger any rules due to not having any prescriptions or lab tests associated.

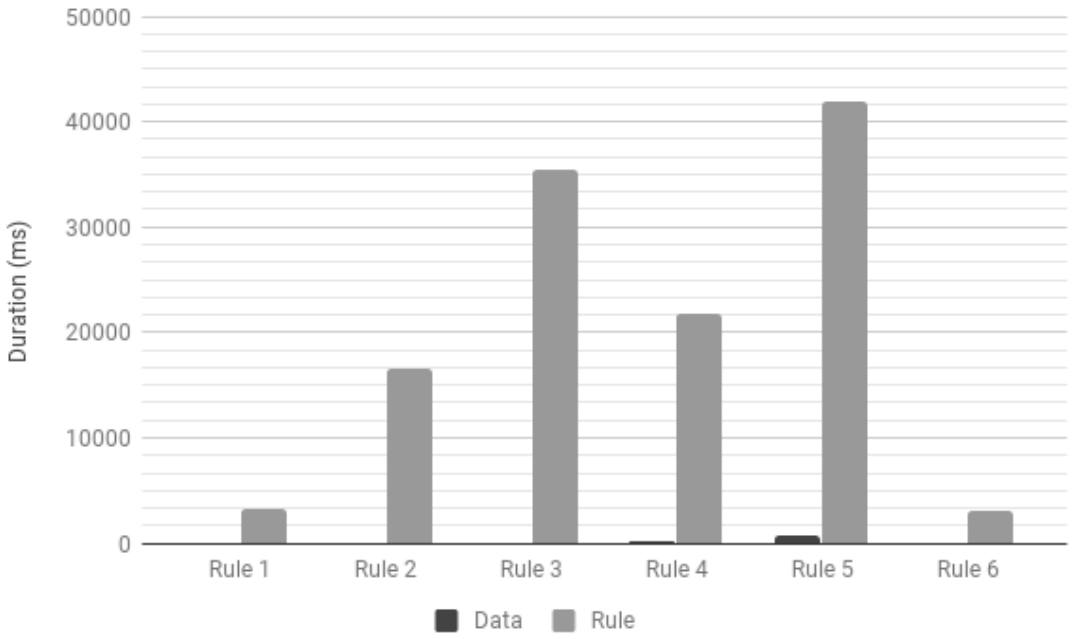

Figure 6.5: Rule and data analyses duration comparison (in ms)

6.3.5 Mule ESB overhead

The graph represented by figure 6.6 compares aggregation times of rule analyses when ran through the Java driver called by a component in the Mule ESB and when ran directly through the MongoDB shell. Running Mule ESB at the same time as the MongoDB could contribute to longer analyses, as well as calling the Java driver through Mule could introduce some kind of overhead. Yet, the aggregation times are hardly different (varying from -5% to 20%) and two of the aggregations were in fact faster when using Mule. For these reasons, there are no correlations that could be inferred regarding Mule aggravating aggregation times.

6.3.6 Scaling

Scaling affected rule analysis the most. With a million documents, analysis took on average 100 more times than when ran with only 5 thousand. The worst case was for rule 5, which analysis' duration increased by a factor of 174 times.

Data analysis was also affected by scaling, although not in the same way that rule analysis was. On average, data analysis durations were 20 times longer, the worst case also being rule 5 with a 69 factor. Oddly, the first rule took less time to be analyzed with a million documents than with 5 thousand. This can be explained by the fact that it was always the first rule to be analyzed, which, as previously mentioned, could introduce some overhead created by any kind of initiation processes in the Mule ESB or even MongoDB.

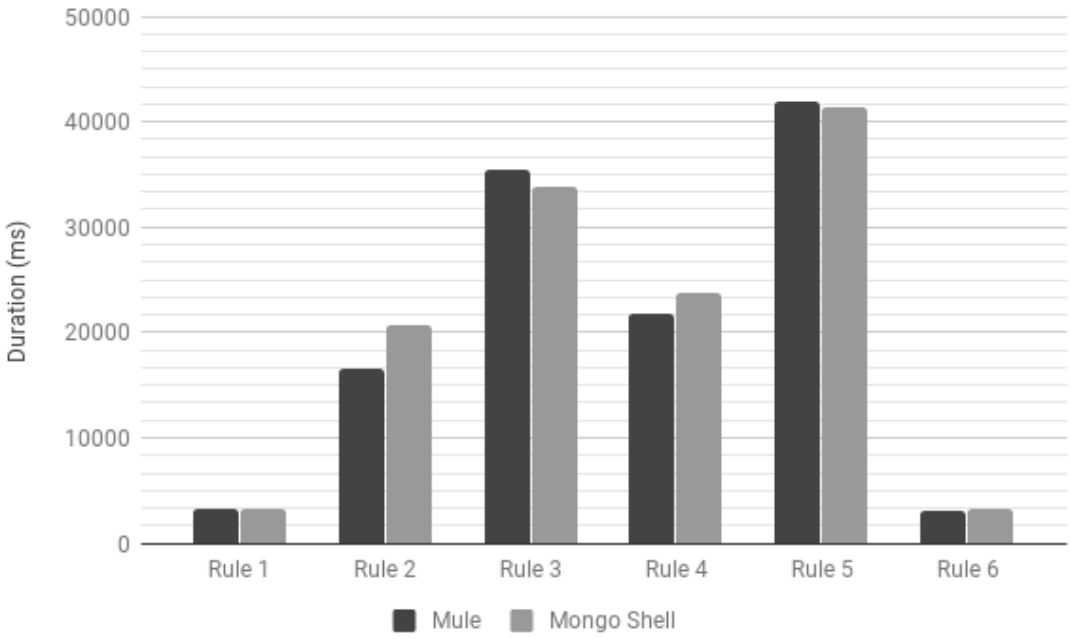

Figure 6.6: Rule analysis duration through Mule and Mongo shell comparison (in ms)

Overall, this test has shown that while data analysis is being performed quickly enough independently of the amount of data in the database, and the notifications being create timely, meeting the requirement of five seconds, rule analysis durations are far from what is expected.

6.3.7 Data analysis distribution

Through the study of graph in the figure 6.9, it can be seen that, as expected, the analysis itself takes more time than the remaining phases of the overall process. This supports the Mule overhead test. Although, figure 6.10, that further divides the Analyzer phase, shows that there are some other phases that, in some cases, are taking as much time as the Finding rules and Analyzing rules (that represents the MongoDB aggregation) stages.

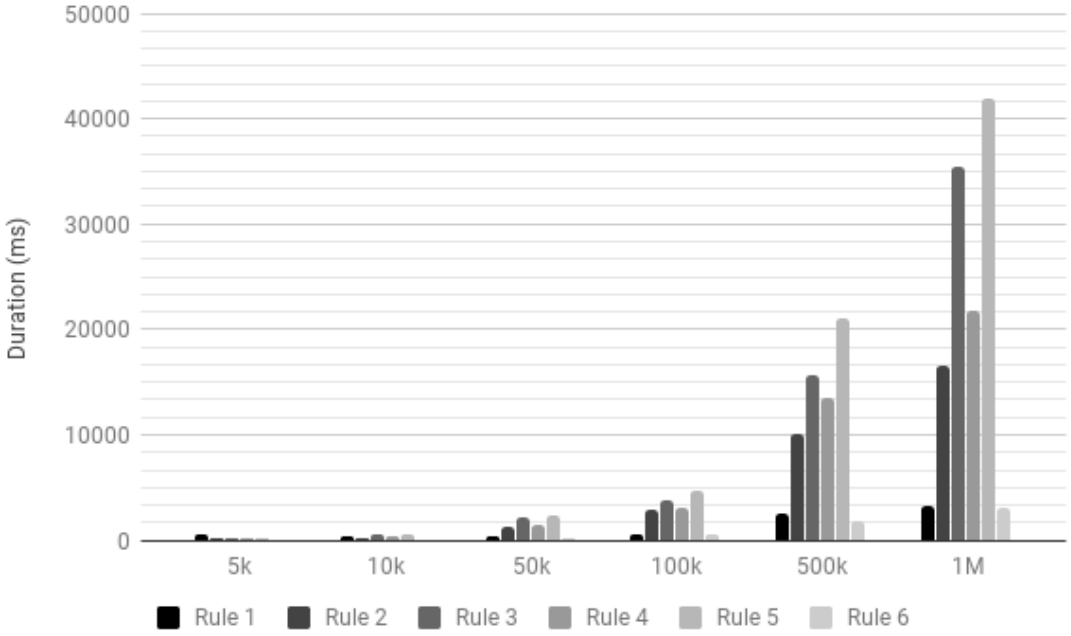

Figure 6.7: Rule analysis duration with increasing data (in ms)

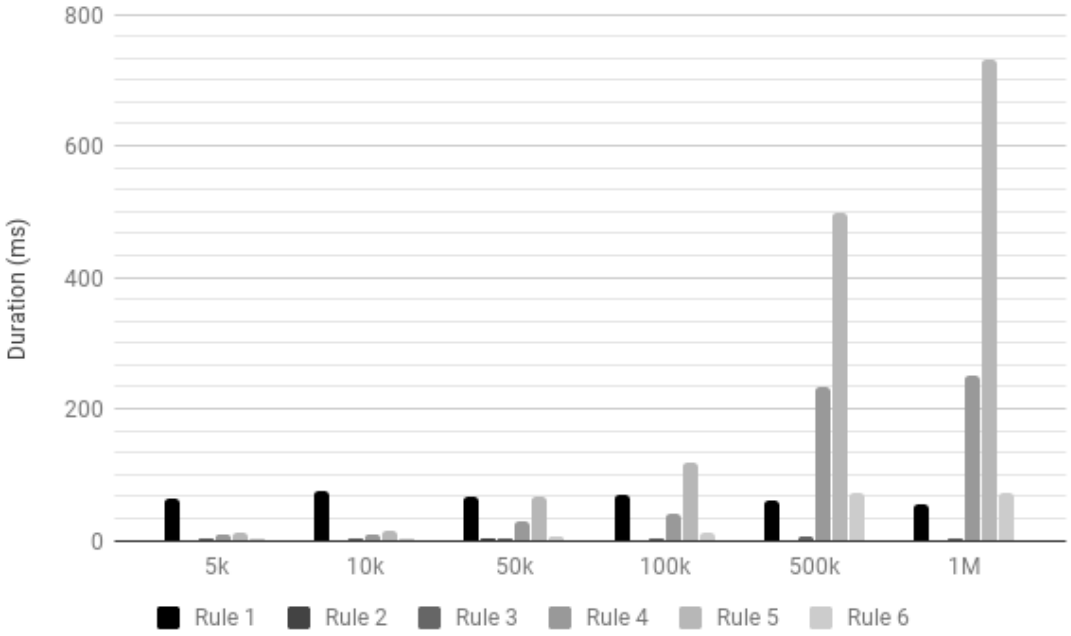

Figure 6.8: Data analysis duration with increasing data (in ms)

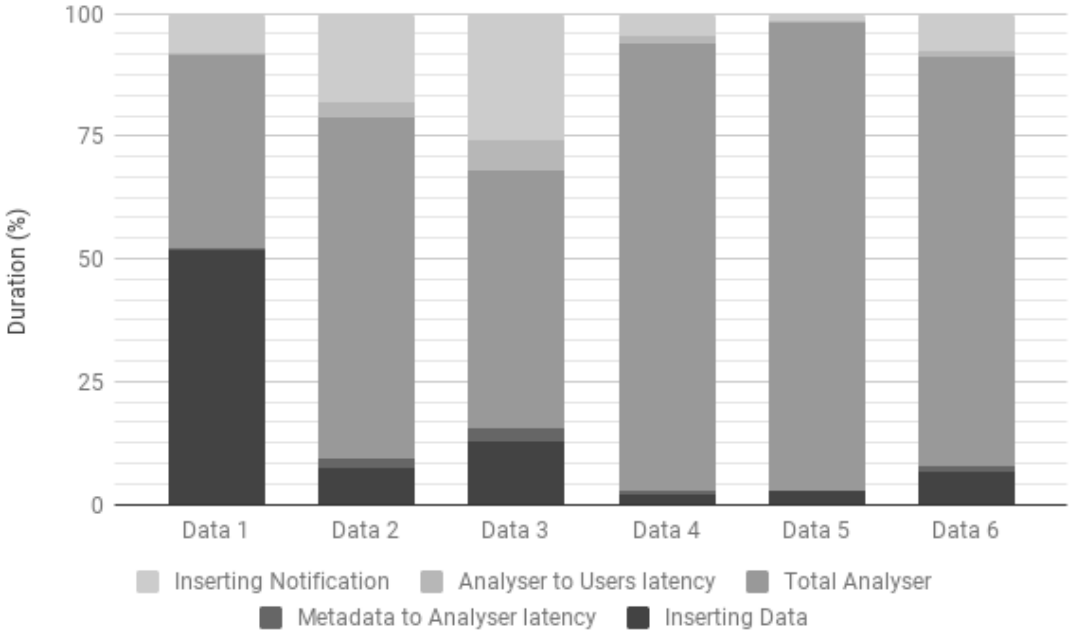

Figure 6.9: Data analysis duration distributed along its different phases (in %)

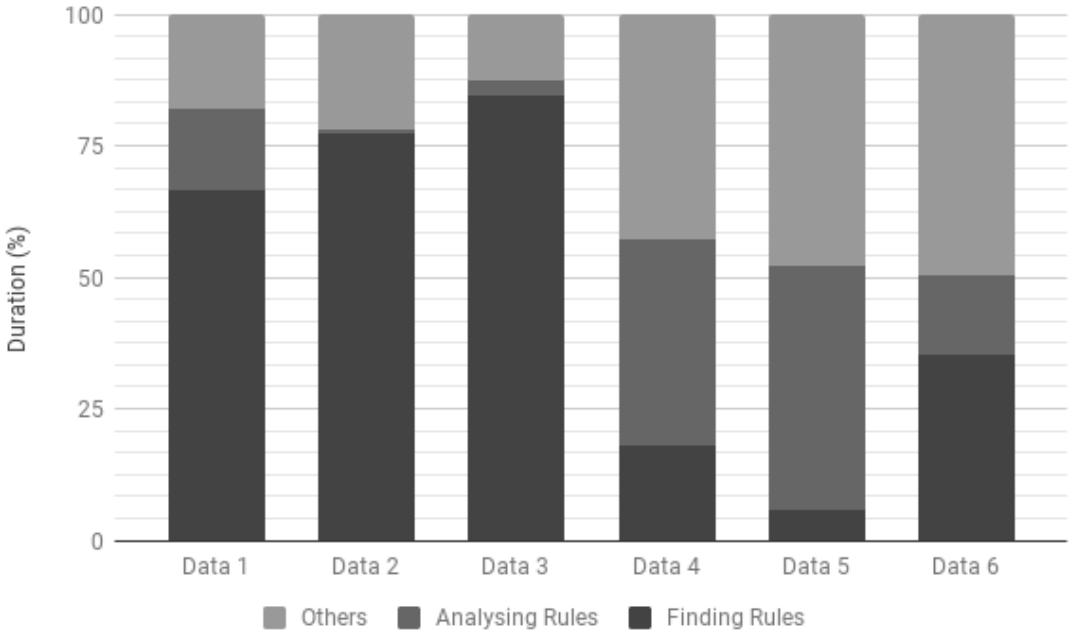

Figure 6.10: Total Analyzer duration distributed along its different phases (in %)

Chapter 7

Conclusions

The main objective of this dissertation was to implement a generic Rule Engine based Notification system, or more specifically, a system that compares user-created rules to data records, both built according to custom meta-data, and to send notifications whenever there is a match. Later, the implemented solution needed to be tested to see whether or not it could be applied in a critical results notification environment, meaning the notification triggering process could not take too much time to allow the responsible health-care professional to react and take action immediately and avoid causing any possible unnecessary damage to a patient.

The implemented solution successfully provides all the required features:

1. agents are able to register meta-data and, afterwards, the corresponding data;
2. users can use this meta-data to build rules;
3. upon data creation, it is analyzed, i.e. compared against the rules,
4. if a match is found, notifications (including a custom description and the data record) are created and sent.

Another major goal was the analysis of 'temporary' data and rules. This one was, as well, accomplished. Users can test a rule and see if any data matches it, or send some data and see if any existing rule includes it in its analysis results.

An essential requirement was the promptness of the whole system, but, more specifically, of the analyses of rules and data. Although the total notification creation time, from receiving the data document, including its analysis and to the notification registration, is below the limit of five seconds, the rule analysis variant should be faster. Its average duration when performed with a million data records of 20 seconds is way above that limit.

Unfortunately, other secondary goals were not achieved.

A webapp is necessary to more easily illustrate how an user could create a rule and how received notifications could be seen. The structure that would run under this app is implemented,

meaning all data access and manipulation operations, from registering and authenticating an user, including searching for meta-data (to aid in creating a rule), to getting the generated notifications. All these operations are indeed implemented and functioning correctly. The missing part is the visual application that would call those operations.

Also, notifications were supposed to be transmitted by at least two different channels or means (e-mail, text message, push notification or webapp) and to any available contact in the matched data. Currently, notifications are only added to a database to be seen by the user who created the rule that triggered the notification.

7.1 Future work

Since not every goal initially proposed was achieved, part of the future work would be to implement the missing features and, then, find ways to improve performance and, as a consequence, reducing analysis time.

Regarding performance improvement, one way this could be achieved is using Elasticsearch [IPa]. Elasticsearch, known for providing a schema-free, document-oriented and exceptionally fast search-engine, could be used in an architecture similar to the one in figure 7.1, used by Auth0 [IPb]. Elasticsearch would act as a replica of the Rule Engine database in MongoDB, applying every modification, it being a document creation, update or deletion, to both databases. Then simple queries, those that don't take too much time, such as searching for meta-data, data or rules could still be ran in MongoDB, but performance-heavy aggregations, as rule and data analysis fundamentally are, would be performed in Elasticsearch. Its integration wouldn't be particularly difficult either. Although Mule ESB currently has no out-of-the-box Elasticsearch connector, interacting with it wouldn't be too demanding, since it provides an HTTP RESTful API to execute all CRUD (create, read, update and delete) operations through the regular HTTP methods, POST, GET, PUT and DELETE, respectively.

Another way to accomplish better analyses performance is through dynamic indexes. This would involve gathering statistics related to data documents, mainly which agent and meta-data and which properties of data are being used the most and then adapt or create indexes according to those statistics. The first step towards implementing such a solution would be to try and use a npm package called Mongo Dynamic Indexer [gen], that provides the mentioned feature. After its deployment, testing would be needed to establish whether the created indexes were actually being used and if they reduce, in fact, aggregation execution times.

There is yet another, and simple, way to use statistic data to improve performance. Currently, the aggregations generated for each Rule are being built with the same order as the conditions appear in. But sorting conditions by the number of registered data documents of the meta-data they make use of, in ascending order, might reduce the aggregation time by discarding failed matches earlier.

During the solution development, some ideas for valuable features surfaced up.

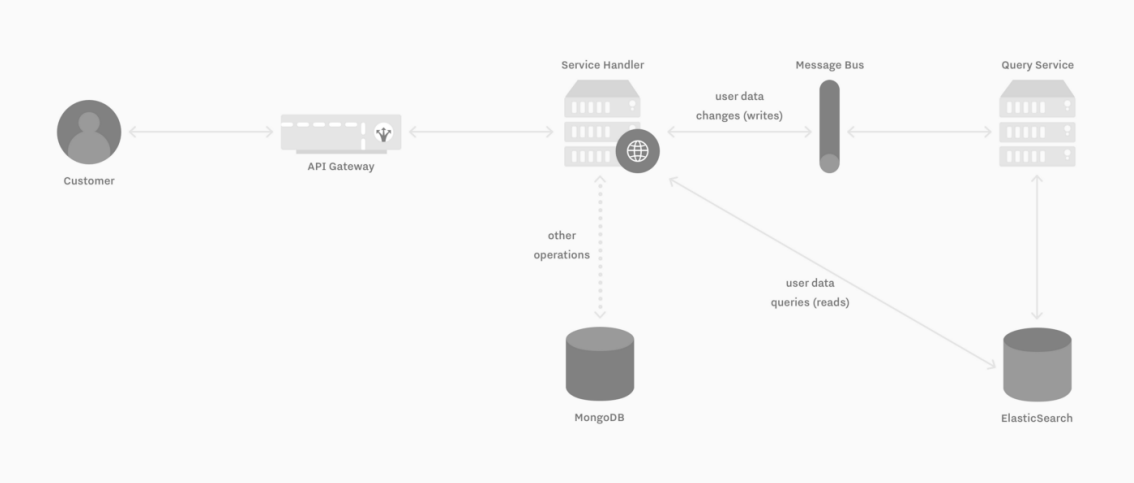

Figure 7.1: Architecture with Mongo and Elasticsearch used in [IPb]

One was to allow agents to declare functions or operators to operate on data documents and output some value. Two scenario examples that demonstrate how useful this can be are:

1. an agent that sends people data, including their date of birth, creating an operator that calculates someone's age,

2. an agent that registers results of Lipid panels, declaring a function to convert HDL cholesterol results between units mg/dL and mmol/L.

Another useful feature to add is the ability to compare values across different conditions. For instance, a doctor might want to be notified if a patient is going to do a lab test after being prescribed some medication. This would require to compare both the prescription date and the test date. Implementing this feature would require adding other variables to the 'let' property of the lookup stages.

The most ambitious feature is to, after creating a webapp, add a Natural Language variant to the rule building. Using Natural-language processing would allow users to simply type the rule they wanted to create, hopefully improving user-experience.

7.2 Final conclusions

Reporting Critical Results is still a serious problem all around the world, putting at risk many patient's health. Existing solutions, though promising, always have some kind of limitation or restriction.

The system here proposed, the Rule Engine based Notification System, though generic, seems to be a viable solution when applied to the Critical Results Notification issue. By studying the tests done to the system, one can see that the results are mostly positive, since the notifications are being generated timely and promptly, as the nature of critical results requires.

Therefore, by publishing this dissertation, the author hopes that the system may solve, or at least may be used as inspiration or foundation to another solution to communicating Critical Results, and ultimately, when implemented in a real health-care environment, it improves patient's quality of care and safety.

References

[Act] Apache ActiveMQ. Apache ActiveMQ. Available at http://activemq.apache.org/.

[Bro98] S J Brown. Multiple patient monitoring system for proactive health management, 1998.

[Cam] Apache Camel. Apache Camel. Available at http://camel.apache.org/.

[Cas] Apache Cassandra. Cassandra. Available at http://cassandra.apache.org/.

[Doc] Docker. Docker - get started. Available at https://docs.docker.com/get-started/.

[Fra16] H. Frank Cervone. Perspectives on informatics in the health sciences for information professionals. 32(4):226–231, 2016.

[gen] genixpro. Mongo Dynamic Indexer. Available at https://www.npmjs.com/package/mongo-dynamic-indexer.

[Gli] Glintt. About Glintt. Available at http://www.glintt.com/en/who-we-are/aboutglintt/Pages/default.aspx.

[Her09] William Hersh. A stimulus to define informatics and health information technology. *BMC Medical Informatics and Decision Making*, 9(1):24, 2009.

[HIM] HIMSS. Electronic Health Records.

[HL7] HL7. Medicationrequest - fhir 3. Available at http://www.hl7.org/fhir/medicationrequest.html.

[IPa] Sebastian Iacomuzzi and Sebastian Peyrott. Elasticsearch: RESTful, Distributed Search & Analytics. Available at https://www.elastic.co/products/elasticsearch3.

[IPb] Sebastian Iacomuzzi and Sebastian Peyrott. From slow queries to over-the-top performance: how Elasticsearch helped us scale. Available at https://auth0.engineering/from-slow-queries-to-over-the-top-performance-how-elasticsearch-helped-us-scale-4fe72ffcb823.

[LOA+14] Ronilda Lacson, Stacy D O'Connor, Katherine P Andriole, Luciano M Prevedello, and Ramin Khorasani. Automated Critical Test Result Notification System: Architecture, Design, and Assessment of Provider Satisfaction. *AJR. American journal of roentgenology*, 203(5):W491–W496, nov 2014.

[Lun72] G.D. Lundberg. When to panic over abnormal values. *MLO Med Lab Obs*, 4:47–54, 1972. cited By 121.

[MM] Ross Mason and Joe McKendrick. The Rising Value of APIs: MuleSoft's digital transformation predictions. Technical report, MuleSoft. Available at https://www.mulesoft.com/ty/wp/rising-value-apis.

[Mona] MongoDB. MongoDB. Available at https://www.mongodb.com/.

[Monb] MongoDB. MongoDB - Aggregation. Available at https://docs.mongodb.com/manual/aggregation/.

[Monc] MongoDB. MongoDB - Indexes. Available at https://docs.mongodb.com/manual/indexes/.

[Mond] MongoDB. MongoDB - Partial Indexes. Available at https://docs.mongodb.com/manual/core/index-partial/.

[MSR12] Raphael W. Majeed, Mark R. Stöhr, and Rainer Röhrig. Proactive authenticated notifications for health practitioners: Two way human computer interaction through phone. volume 180, pages 388–392, 2012.

[Mula] API-led Connectivity: The Next Step in the Evolution of SOA. Technical report, MuleSoft. Available at https://www.mulesoft.com/ty/wp/api-led-connectivity.

[Mulb] Mulesoft. Anypoint Platform. Available at https://www.mulesoft.com/platform/enterprise-integration.

[Pea] Shana Pearlman. What is API-led Connectivity? Available at https://blogs.mulesoft.com/dev/api-dev/what-is-api-led-connectivity/.

[PSPP17] Elisa Piva, Laura Sciacovelli, Michela Pelloso, and Mario Plebani. Performance specifications of critical results management. *Clinical Biochemistry*, 50(10-11), 2017.

[Rab] RabbitMQ. RabbitMQ. Available at https://www.rabbitmq.com/.

[Red] Redis. Redis. Available at https://redis.io/.

[RNF+17] Erika M. Reese, Randin C. Nelson, Willy A. Flegel, Karen M. Byrne, and Garrett S. Booth. Critical value reporting in transfusion medicinea survey of communication practices in us facilities. *American Journal of Clinical Pathology*, 147(5):492–499, 2017.

[Sir05] Ronald L. Sirota. Error and error reduction in pathology. *Archives of Pathology & Laboratory Medicine*, 129(10):1228–1233, 2005. PMID: 16196509.

Appendix A

Test rules documents

This appendix presents the documents used to build each rule during the testing phase, as mentioned in chapter 6.

```
{
  "description": ">=80 years;MALE;Aspirin",
  "output": "<person.name> was prescribed <prescription.medication>!",
  "conditions": [
    {
      "agentID": "5a5fe5298a2b36edb431f6e1",
      "metadata": "person",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "GTE",
            "property": "age",
            "value": 80
          },
          {
            "operator": "EQ",
            "property": "gender",
            "value": "MALE"
          }
        ]
      }
    },
    {
      "agentID": "5a3935810c19c6068c3c9624",
      "metadata": "prescription",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "EQ",
            "property": "ingredient",
            "value": "Aspirin"
          }
        ]
      }
    }
  ],
  "joins": [
    [ "id", "patient" ]
  ]
}
```

Listing A.1: Document (abbreviated) for the first Rule used in the tests.

```
{
  "description": "FEMALE;Lipid Panel",
  "output": "<person.name> did a Lipid Panel!",
  "conditions": [
    {
      "agentID": "5a5fe5298a2b36edb431f6e1",
      "metadata": "person",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "EQ",
            "property": "gender",
            "value": "FEMALE"
          }
        ]
      }
    },
    {
      "agentID": "5a5fe52b8a2b36edb431f6e2",
      "metadata": "labTest",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "EQ",
            "property": "test",
            "value": "Lipid panel"
          }
        ]
      }
    }
  ],
  "joins": [
    [ "id", "patient" ]
  ]
}
```

Listing A.2: Document (abbreviated) for the second Rule used in the tests.

```
{
  "description": ">20, <50 years;Aspirin|Ibuprofen",
  "output": "<person.name> was prescribed <prescription.medication>!",
  "conditions": [
    {
      "agentID": "5a5fe5298a2b36edb431f6e1",
      "metadata": "person",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "GT",
            "property": "age",
            "value": 20
          },
          {
            "operator": "LT",
            "property": "age",
            "value": 50
          }
        ]
      }
    },
    {
      "agentID": "5a3935810c19c6068c3c9624",
      "metadata": "prescription",
      "filter": {
        "operator": "OR",
        "expressions": [
          {
            "operator": "EQ",
            "property": "ingredient",
            "value": "Aspirin"
          },
          {
            "operator": "EQ",
            "property": "ingredient",
            "value": "Ibuprofen"
          }
        ]
      }
    }
  ],
  "joins": [
    [ "id", "patient" ]
  ]
}
```

Listing A.3: Document (abbreviated) for the third Rule used in the tests.

```
{
  "description": "MALE;Desloratadine&Aerosol;Urinalisys",
  "output": "<person.name> was prescribed <prescription.medication> and did a
      Urinalysis!",
  "conditions": [
    {
      "agentID": "5a5fe5298a2b36edb431f6e1",
      "metadata": "person",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "EQ",
            "property": "gender",
            "value": "MALE"
          }
        ]
      }
    },
    {
      "agentID": "5a3935810c19c6068c3c9624",
      "metadata": "prescription",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "EQ",
            "property": "ingredient",
            "value": "Desloratadine"
          },
          {
            "operator": "EQ",
            "property": "form",
            "value": "Aerosol"
          }
        ]
      }
    },
    {
      "agentID": "5a5fe52b8a2b36edb431f6e2",
      "metadata": "labTest",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "EQ",
            "property": "test",
            "value": "Urinalysis"
          }
        ]
      }
    }
  ],
  "joins": [ [ "id", "patient" ], [ "id", "patient" ] ]
}
```

Listing A.4: Document (abbreviated) for the fourth Rule used in the tests.

```
{
  "output": "<person.name> takes <prescription.medication> and did a Urinalysis!",
  "conditions": [
    {
      "metadata": "person",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "GTE",
            "property": "age",
            "value": 45
          },
          {
            "operator": "EQ",
            "property": "gender",
            "value": "FEMALE"
          }
        ]
      }
    },
    {
      "metadata": "prescription",
      "filter": {
        "operator": "OR",
        "expressions": [
          {
            "operator": "EQ",
            "property": "form",
            "value": "Pills"
          },
          {
            "operator": "EQ",
            "property": "form",
            "value": "Syrup"
          }
        ]
      }
    },
    {
      "metadata": "labTest",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "EQ",
            "property": "test",
            "value": "Urinalysis"
          }
        ]
      }
    }
  ],
  "joins": [ [ "id", "patient" ], [ "id", "patient" ] ]
}
```

Listing A.5: Document (abbreviated) for the fifth Rule used in the tests.

```
{
  "output": "<person.name> takes <prescription.medication> and did a Urinalysis!",
  "conditions": [ {
      "metadata": "person",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "LT", "property": "age", "value": 40
          },
          {
            "operator": "EQ", "property": "gender", "value": "FEMALE"
          }
        ]
      }
    }, {
      "metadata": "prescription",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "EQ", "property": "ingredient", "value": "Etofenamate"
          },
          {
            "operator": "GTE", "property": "prescDate", "value": "2017-11-21T00
                :00:00"
          }
        ]
      }
    }, {
      "metadata": "labTest",
      "filter": {
        "operator": "AND",
        "expressions": [
          {
            "operator": "EQ", "property": "test", "value": "Complete blood count"
          },
          {
            "operator": "GTE", "property": "reqDate", "value": "2017-11-21T00:00:00
                "
          }
        ]
      }
    }
  ],
  "joins": [ [ "id", "patient" ], [ "id", "patient" ] ]
}
```

Listing A.6: Document (abbreviated) for the sixth Rule used in the tests.

Appendix B

Test results data tables

This appendix presents the data produced in the tests and represented in the tables in chapter 6.

Table B.1: Rule Analysis duration with and without $sort and $limit stages comparison (in ms)

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**
With stages	3297	16606	35380	21851	41913	3162
Without stages	4936	17413	35783	23288	42627	5535

Table B.2: Data Analysis duration with and without $sort and $limit stages comparison (in ms)

	Data 1	**Data 2**	**Data 3**	**Data 4**	**Data 5**	**Data 6**
With stages	56	2	4	251	731	73
Without stages	61	2	3	496	988	128

Table B.3: Rule Analysis duration with and without indexes comparison (in ms)

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**
With indexes	3297	16606	35380	21851	41913	3162
Without indexes	5494	43087	41736	79936	62016	28472

Table B.4: Data Analysis duration with and without indexes comparison (in ms)

	Data 1	**Data 2**	**Data 3**	**Data 4**	**Data 5**	**Data 6**
With indexes	56	2	4	251	731	73
Without indexes	962	337	548	1024	1329	711

Table B.5: Rule and Data Analyses duration comparison (in ms)

	Rule/Data 1	**Rule/Data 2**	**Rule/Data 3**	**Rule/Data 4**	**Rule/Data 5**	**Rule/Data 6**
Rules	3297	16606	35380	21851	41913	3162
Data	56	2	4	251	731	73

Table B.6: Rule Analysis duration with increasing Data (in ms)

# Data	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**
5k	610	182	250	181	241	48
10k	293	225	489	339	490	66
50k	437	1354	2259	1502	2441	284
100k	595	2839	3749	3011	4646	477
500k	2496	10053	15622	13552	21130	1835
1M	3297	16606	35380	21851	41913	3162

Table B.7: Data Analysis duration with increasing Data (in ms)

# Data	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**
5k	63	2	3	9	11	4
10k	75	2	4	9	14	4
50k	67	2	4	29	65	7
100k	69	1	4	42	119	12
500k	61	1	7	232	498	72
1M	56	2	4	251	731	73

Table B.8: Data Analysis distributed along its different phases (in ms)

	Insert Data	**Metadata to Analyser**	**Total Analyser**	**Analyser to Users**	**Insert Notification**	**Total**
Data 1	1430	18	1092	9	223	2772
Data 2	33	9	312	13	82	448
Data 3	71	15	293	34	143	556
Data 4	28	7	1278	17	67	1397
Data 5	89	7	3152	12	50	3309
Data 6	40	6	487	6	45	583

Table B.9: Total Analyser duration distributed along its different phases (in ms)

	Finding Rules	**Analysing Rules**	**Others**	**Total Analyser**
Data 1	728	169	195	1092
Data 2	241	2	69	312
Data 3	248	8	37	293
Data 4	229	502	547	1278
Data 5	183	1463	1506	3152
Data 6	172	73	242	487

Table B.10: Rule Analysis duration through Mule and Mongo Shell comparison (in ms)

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**
Mule (Java)	3297	16606	35380	21851	41913	3162
Mongo Shell	3179	20700	33855	23835	41391	3260