HL7v2 Tools for Modelling and Testing

Ana Gadelho
FEUP — FER UniZG
Porto, Portugal — Zagreb, Croatia
up201806309@up.pt
ana.gadelho@fer.hr

André Gomes
FEUP — FER UniZG
Porto, Portugal — Zagreb, Croatia
up201806224@up.pt
andre.gomes@fer.hr

Julien Goethal
IPSA — FER UniZG
Toulouse, France — Zagreb, Croatia
julien.goethal@ipsa.fr
julien.goethal@fer.hr

***Abstract*—HL7v2 is a widely used standard for exchanging healthcare data, but it can be challenging to work with due to its complexity. To simplify the process of working with HL7v2 messages, various tools and technologies have been developed. In this seminar, we will explore two of these tools: HAPI and the Google Cloud Healthcare API. We will be discussing the limitations of each and comparing them. This seminar will be useful for gaining a deeper understanding of the capabilities of these tools and how they can help with modelling, testing, and integrating HL7v2 data into healthcare applications.**

***Index Terms*—HL7, HL7v2, HAPI, ADT messages, Google Cloud Healthcare API**

I. Introduction

Being able to send messages in a secure, efficient and reliable way is a crucial topic in the scope of IT for Healthcare since there's a need to keep patients' data private but still be able to send information between healthcare units and systems. Having this in mind, it's of great importance to have a defined standard for communication in healthcare institutions that is widely accepted. HL7 is one of these standards. This report aims to better understand the use of the health message standard HL7v2 and explore tools for modelling and testing HL7v2 messages, stating their advantages and drawbacks.

To better introduce the topic, HL7v2 is explained, then the researched tools that implement this protocol are stated, and its features are explained. After that, the experiments we carried out are described, and the results are shown, leading to the conclusions extracted.

II. HL7 Messaging Standard

Health Level 7 (HL7)[1] is a standards-setting organization. They have developed communication protocols widely used in the United States that have been recognized and implemented internationally. Its mission is to provide standards for the exchange, management, and integration of data that support clinical patient care and the administration, delivery, and evaluation of healthcare services. [2] Shared reference models of the healthcare and technical domains unify the HL7 specifications.

A. HL7v2

HL7v2 is a version of the HL7 standard for exchanging electronic health record data. It is a widely used standard in the healthcare industry for exchanging clinical and administrative data between systems, such as electronic health records (EHRs), practice management systems, and laboratory information systems. HL7v2 has been in use for many years and is a well-established standard in the healthcare industry. However, it has some limitations, including being inflexible and difficult to implement.

Older versions of HL7 include HL7v1 and HL7v0. HL7v1 was the first version of the HL7 standard and was released in the 1980s. It was primarily used to exchange laboratory data and had a limited scope. HL7v0 was an earlier version of the HL7 standard that was used for the exchange of administrative data.

Compared to these older versions, HL7v2 is a more comprehensive standard that covers a wider range of healthcare information, including clinical and administrative data. It is also more widely used and has become a de facto standard in the healthcare industry.

In recent years, HL7 has released newer versions of the standard, including HL7v3 and HL7 FHIR (Fast Healthcare Interoperability Resources). These newer versions address some of the limitations of HL7v2 and are intended to make it easier to exchange healthcare data between systems. HL7v3 is a more comprehensive standard that covers a wider range of healthcare information than HL7v2. It is based on a set of principles and guidelines for developing interoperable healthcare information systems. FHIR is a newer standard that is based on modern web standards and is intended to be easier to implement and use than previous versions of HL7. It is designed to be flexible and adaptable, making it easier to integrate with different systems and technologies.

B. Types of messages

HL7v2 supports a wide variety of messages that are used in the healthcare industry.

A few examples of important messages are the following:

- **Admission, Discharge, and Transfer (ADT) messages:** This message is used to communicate information about a patient's admission, discharge, or transfer to a healthcare facility. It includes patient demographic information, such as the patient's name, date of birth, and medical record number.
- **Order messages:** This message is used to communicate orders for lab tests, medications, or other clinical procedures. It includes the patient's medical record number, the order details, and any relevant notes or instructions.

- **Results messages:** This message is used to communicate test results or other clinical findings to the ordering provider or to the patient's electronic health record. It includes the patient's medical record number, the test results, and any relevant notes or comments.
- **Patient Demographic messages:** This message is used to communicate information about a patient's demographic details, such as their name, address, and contact information.
- **Payment Advice messages:** This message is used to communicate information about a patient's payment for healthcare services, including the amount due, the payment method, and any relevant notes or comments.

C. Possible Errors

Several types of errors can occur in the exchange of HL7v2 messages. Some of the most common errors we should look out for in our experiments include the following:

- **Syntax errors:** These occur when the HL7 message does not conform to the HL7 v2.5.1 standard. For example, a field may be missing or a segment may be in the wrong order.
- **Semantic errors:** These occur when the HL7 message is syntactically correct, but the information contained within the message is incorrect or inconsistent. For example, a patient's name may be misspelt, or their birthdate may be incorrect.
- **Data type errors:** These occur when the data in a field is not of the correct type. For example, if a field is supposed to contain a date, but it contains a string of text instead.
- **Data value errors:** These occur when the data in a field is not in the correct range or format. For example, if a field is supposed to contain a date and it contains a value that is not a valid date.
- **Code set errors:** These occur when the data in a field is not valid according to the code set specified in the HL7 standard.
- **Security errors:** These occur when the message exchange is not secure. For example, when messages are not encrypted and/or authenticated or if the messages are intercepted by an unauthorized party.

It is important to have these errors in mind when we experiment with the testing platforms in order to see how they are handled.

D. Messages Acknowledgment

In HL7v2, acknowledgement (ACK) messages are used to confirm the reception of a message by the receiving system. There are two types of acknowledgement messages in HL7v2: original and enhanced.

Original acknowledgement is the default type of acknowledgement message used. It simply confirms that the message has been received by the receiving system.

An Enhanced acknowledgement message is an extension of the original acknowledgement message. It provides additional information about the status of the message and any errors that may have occurred during processing. Enhanced ACKs can be used to provide detailed information about why a message was rejected or why it could not be processed.

It's worth noting that the enhanced ACK message provides a more detailed response to the sender regarding the status of the message. This makes it more useful for troubleshooting and debugging, especially in large and complex systems.

III. Existing tools

A. HAPI

HAPI (Health Level 7 Application Programming Interface)[3] is an open-source Java library for parsing and creating HL7 messages. It was developed by the Health Level 7 (HL7) organization to provide a simple and easy-to-use interface for developers to work with HL7 messages.

HAPI provides a set of classes and methods for parsing HL7 messages, creating new messages, and validating messages against the HL7 standard. It also includes support for various HL7 message versions, including HL7v2 and HL7 FHIR (Fast Healthcare Interoperability Resources).

It is widely used in the healthcare industry for applications that need to exchange HL7 messages, such as electronic health records (EHRs), practice management systems, and laboratory information systems. It is a popular choice for developers because of its ease of use and comprehensive support for HL7 message parsing and creation.

B. Google Cloud Healthcare API

Google Cloud's Healthcare API[4] is a cloud-based platform that provides tools and services for storing, processing, and analyzing healthcare data. It supports data exchange using the HL7 standard, including the ability to parse and create HL7 messages.

The Healthcare API includes a set of APIs and libraries for working with HL7 data, including the ability to:

- Import and export HL7 messages
- Transform HL7 messages into different formats
- Validate HL7 messages against the HL7 standard
- Search and query HL7 data

Google Cloud's Healthcare API is designed to enable the integration and interoperability of healthcare data across different systems and organizations. It is used by healthcare providers, payers, life sciences companies, and other organizations to manage and analyze healthcare data.

IV. Experimental Results

Show the results and comparisons made

A. HAPI

1) Setup: Since HAPI is a JAVA API, it is first necessary that the working machine has a JDK available to compile and run Java code.

HAPI can be used in one of the ways. Either inside a Maven project by adding to the `pom.xml` the core HAPI library and the specific version library for the data structures or by downloading the `.jar` files of the source code containing

also the core HAPI library and the data structures, but some additional HAPI dependencies like SLF4J and LOG4J for logging.

In terms of code, every execution needs someplace to start, and with HAPI, that is the singleton `HapiContext`, normally initialized with `new DefaultHapiContext()`.

2) Communication Protocol: Most HL7 systems communicate messages with one another using the Minimum Lower Layer Protocol (often shortened to "MLLP"). MLLP can be considered a wrapper protocol on top of the TCP/IP protocol since the communicating systems need to be able to recognize the start and the end of each message (i.e. headers and trailers) as TCP/IP data is simply a stream of continuous bytes. MLLP typically uses non-printable characters to serve as wrapping characters around the core HL7 message information exchanged between these systems. If one were programming an HL7 system from scratch, then one would need to be familiar with many aspects of network programming, such as sockets, to accomplish the task of coordinating the steps required to establish TCP/IP communications between the two systems involved. Luckily, HAPI abstracts that and provides some helpful classes to enable message exchange. The following are some of the features provided for HL7 message communication by HAPI:

- Transmit HL7 messages over the MLLP protocol
- Transmit HL7 messages through a serialized file, email, etc
- Transmit HL7 messages through HTTP (and HTTPS) using the HoH specification (stands for "HL7 over HTTP")
- Transmit HL7 messages over a Restful Web Service Interface (using the HAPI HL7 FHIR Tooling)

3) Message Creation: Before sending a message, it is necessary to create one, in this case, an instantiation of the `Message` class. If there is a message in string format, a `Parser` can be instantiated from the `HapiContext` in order to process the string into a `Message`, which can then be cast into a specific type, depending on the message, for example, a `ADT_A01` message,

Messages can also be created from scratch by starting with a `new ADT_A01();`, for example, and adding information to all of the required fields.

4) Sending Messages: To send a message, there must be a receiver active. Before going in-depth about how to create a receiver programmatically, HAPI provides a testing tool called `HAPI TestPanel` to receive messages and simplify the testing process.

To set up `HAPI TestPanel` for receiving messages, first, the compiled `.jar` or source code should be downloaded, and the executable `.jar` file can be run. In the program, a message listener can be created, and it only requires one parameter, the port number. After setting this up, an HL7 listener will be enabled on the machine.

On the sending side, two objects need to be created: a `Connection` using the `HapiContext` by calling the method `newClient` and passing the arguments `address` and `portNumber`; and after that a `Initiator` by calling `connection.getInitiator()`. Using the initiator and method `sendAndReceive`, it is possible to send the `Message` through the parameters and get the response as the return value. The `HAPI TestPanel` will respond with simplified ACK messages.

5) Receiving Messages: An application to receive messages can also be created using the HAPI toolkit. For this, instead of a `Connection` and a `Initiator`, the receiving application will need a `HL7Service`, obtained via `HapiContext.newServer()` and with the only argument being `portNumber`. the service/server can be initialized via `startAndWait()` and stopped with `stopAndWait()`.

The responses obtained from here will always be "Application Internal Error" because there aren't any message handlers configured. To add a message handler to the server, it is also necessary to add an application router to know to which handler the message should be forwarded.

For the application router, a class that implements the interface `AppRoutingData` is needed and overrides the following methods:

- `getVersion()`
- `getTriggerEvent()`
- `getProcessingId()` - here the return can be "*" to accept all the ID's
- `getMessageType()`

For the message handler, a class that implements the interface `Receiving Application` is needed and overrides the following methods:

- `canProcess()`
- `processMessage()` - that must return a `Message`

With the above classes defined, after creating the `HL7Service`, the classes can be bound to the served using the method `registerApplication()`. With this, the messages will have responses, and the error message will not be sent anymore.

6) Error Handling: An HL7 listener should be fitted with good exception-handling capabilities as well as solid error-logging and alerting capabilities. The HAPI framework provides plenty of support in these aspects by providing several configuration options and "hooks" to manage exception-related processing during the message flow using primarily three different ways:

- By using a special HL7 exception class
- By using application exception policies
- By using exception handlers that can be configured for connections

Logging what happens during exceptions can also be useful for future fixes. It can be done by implementing `ConnectionListener`'s and registering them to the server.

B. Google Cloud Healthcare API

1) Setup: The Google Cloud Healthcare API provides a way to work with HL7v2 data using the Cloud Healthcare API client library, which is available in various programming

languages, including Java, Python, and Node.js. We chose to use Python for our experiments. Because of that, the Google Cloud Python Client[5] was very useful to us since it had the necessary Python helper functions to use this API.

To use this API, it's needed to set up a project in the Google Cloud Console, enable the Healthcare API, and create credentials for the application.

Once the setup is done, it's possible to use the client library to interact with the API.

For us, this process took some time and involved several steps, but in the end, it was possible to get it done. It's worth mentioning that the Google Cloud Platform provides some comprehensive tutorials on this topic, which significantly helped the process.

2) Communication protocol: The most common protocol in healthcare systems to send messages between different systems and applications is the Minimal Lower Layer Protocol (MLLP). This simple protocol is used to transmit HL7v2 messages over TCP/IP networks. It is widely used because it is a simple and efficient protocol for transmitting HL7v2 messages, but it has some limitations. It doesn't provide functionalities such as message validation or converting messages to other formats. Additionally, it doesn't have built-in security features, so it's important to secure the communication through other means.

The Google Cloud Healthcare API does not use the Minimal Lower Level Protocol (MLLP) to transmit HL7v2 messages. It uses the HTTPS protocol with gRPC as the transport protocol, which is a more efficient and secure protocol than the TCP protocol that MLLP uses. This allows for performing functionalities such as converting HL7v2 messages to FHIR and performing HL7v2-based data validation.

To allow sending and receiving HL7v2 messages over the MLLP protocol, the Google Cloud Platform provides the Google Cloud MLLP Adapter. It works as an adapter between the MLLP-based systems we want to integrate with Google Cloud services and the Google Cloud Healthcare API.

3) Storing messages: In this API, there's the concept of the HL7v2 store, which is a container for HL7v2 messages used to organize and manage HL7v2 messages within the Google Cloud platform. Each HL7v2 store belongs to a specific Google Cloud project and can be used to hold one or more HL7v2 messages.

An HL7v2 store can be thought of as a virtual repository for storing and managing HL7v2 messages. It allows for creating, reading, updating, deleting, and querying messages. Each store has a unique name within the project and is associated with a specific Cloud Healthcare API service instance.

The HL7v2 store also provides additional features, such as message profile management, which allows validating messages against a specific message profile.

The HL7v2 store is a central piece when working with HL7v2 messages in the Google Cloud Healthcare API, and we need to provide the name of an HL7v2 store when sending or receiving messages.

Sending and receiving messages between stores is done by a PUB/SUB pattern, where stores are subscribed to other stores and receive the messages published by those stores.

4) Sending messages: To send a message, we can use the `messages.send` method, that has as arguments the HL7v2 store from where the message is being sent and the content of the message.

5) Receiving messages: To receive a message, we can use the `messages.ingest` method, that has as arguments the HL7v2 store from where the message came and the content of the message.

6) Validating messages: This API automatically validates ingested HL7v2 messages using either the default message schema or a personalized schema that needs to be defined accordingly to the specific needs of the given HL7v2 store.

7) Acknowledgent: The Google Cloud Healthcare API supports enhanced ACK (acknowledgement) messages, which provide more detailed information about the delivery of an HL7v2 message than standard ACK messages.

When an HL7v2 message is sent through the API, the recipient will respond with an enhanced ACK message. The enhanced ACK message will include information such as the acceptance or rejection of the message, the message control ID, and the message type. It will also include a list of segments, each with a code indicating whether it was accepted, rejected, or ignored. This provides more detailed information about the delivery of the message than a standard ACK message, which typically only includes a status code indicating whether the message was successfully received.

The API also supports sending and receiving NACK (negative acknowledgement) messages, which are used to indicate that an error has occurred during the delivery of an HL7v2 message. NACK messages also include detailed information about the error, including a code and a description of the problem.

The enhanced ACK and NACK messages provide additional information that can be used to troubleshoot and fix any issues that may have occurred during the delivery of a message. It also allows for a more robust message-handling process.

8) Conversion to FHIR: The API also allows us to convert HL7v2 messages to FHIR (Fast Healthcare Interoperability Resources) format, which is a more modern, RESTful format for exchanging healthcare data. This is done with the $hl7v2\_to\_fhir$ method, to which the HL7v2 message should be passed as a string.

9) Error Handling: The Google Cloud Healthcare API is designed to catch many types of errors that can occur when working with HL7v2 messages. However, there are a few types of errors that it may not catch:

Syntax errors in the HL7v2 message: The API does provide data validation capabilities, but if the message does not conform to the standard syntax of HL7v2 or is not well-formed, the API may not be able to process the message correctly.

Semantic errors: While the API can be able to validate the syntax of a message, it may not be able to detect semantic errors, such as invalid codes or incorrect data in the message.

Errors related to using the API: If there are issues with authenticating the request or with the configuration of the API, the API may not be able to process the message correctly.

C. Comparisons and Key Findings

In terms of setup, since HAPI can be run on a local machine, the setup process is straightforward if it is detailed and followed correctly. The starting point for a HAPI project is less developed than a starting point for a Google Cloud Healthcare API (GCHAPI), which makes it easier to set up. The GCHAPI has some templates for creating simple HL7v2 testing environments in the Google Cloud Console, but they still need to be properly configured, and this requires quite some knowledge of how this console works. Still, after some work and following the provided tutorials, it was possible to set it up.

HAPI uses, by default, the MLLP protocol, while GCHAPI uses HTTPS with gRPC. But it is worth noting that HAPI also has the feature to send messages through HTTP (and HTTPS), and GCHAPI has a wrapper to send and receive MLLP messages.

HAPI works using the Request-Reply pattern. For each Message that is sent, a reply is returned. Multiple messages can still be sent, but that also entails multiple replies. For GCHAPI the pattern used is Pub-Sub.

The error handling in GCHAPI is already built-in, so the developer does not need to worry about it. For HAPI, there are exception classes that catch the errors and exception handlers can be configured for different connections.

V. Conclusion

In conclusion, HL7v2 is a widely used standard for exchanging healthcare data, but it can be challenging to work with due to its complexity. There are various tools and technologies available that can help with modelling and testing HL7v2 messages, such as the ones we explored: HAPI and Google Cloud Healthcare API.

the Health Level Seven (HL7) Application Programming Interface (HAPI) is a powerful tool that enables developers to easily create and transmit HL7 messages in various healthcare applications. HAPI offers a wide range of features, including support for various HL7 versions, message validation, and customizable parsing options. It also provides a convenient and consistent interface for sending and receiving HL7 messages, which can greatly simplify the development process. Overall, HAPI has the ability to lower the barrier of entry for HL7 integration, increase the interoperability of healthcare systems, and ultimately improve the delivery of healthcare. It is a reliable and valuable solution for implementing HL7 messaging in healthcare applications.

The Google Cloud Healthcare API is a powerful tool for working with HL7v2 data, providing functionalities such as sending and receiving HL7v2 messages, converting them to FHIR, validating messages and handling ACK and NACK messages. It also provides data validation capabilities, message profile management, and it uses the HTTPS protocol to secure the transmission of HL7v2 messages.

It's important to note that HL7v2 has slowly been swapped in favour of its successors (HL7v3 and FHIR). This means that it was a bit difficult to find recent documentation about it, especially the particularities of version 2.5.1 we were told to use.

This seminar made us gain a broader and more profound knowledge of the HL7v2 standard and the tools used, and it was a practical way to learn about the theory we have been studying for this course.

Overall, the availability of tools such as these can help to simplify the process of working with HL7v2 messages, providing a range of functionalities for modelling, testing and integrating HL7v2 data into healthcare applications. As the healthcare industry continues to evolve, it is important to stay up-to-date with the latest tools and technologies available to help with HL7 messaging.

References

[1] *HL7 Standards Product Brief - HL7 Version 2 Product Suite — HL7 International*. URL: https://www.hl7.org/implement/standards/product_brief.cfm?product_id=185 (visited on 12/09/2022).

[2] Robert H. Dolin et al. "The HL7 Clinical Document Architecture". In: *Journal of the American Medical Informatics Association* 8.6 (Nov. 1, 2001), pp. 552–569. ISSN: 1067-5027. DOI: 10.1136/jamia.2001.0080552. URL: https://doi.org/10.1136/jamia.2001.0080552 (visited on 12/11/2022).

[3] *HAPI – The Open Source HL7 API for Java*. URL: https://hapifhir.github.io/hapi-hl7v2/ (visited on 12/09/2022).

[4] *HL7v2 — Cloud Healthcare API*. Google Cloud. URL: https://cloud.google.com/healthcare-api/docs/concepts/hl7v2 (visited on 12/11/2022).

[5] *googleapis/google-cloud-python: Google Cloud Client Library for Python*. https://github.com/googleapis/google-cloud-python#google-cloud-python-client. (Accessed on 01/11/2023).