Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 238 (2024) 920–925

Procedia Computer Science

www.elsevier.com/locate/procedia

The 5th International Workshop on Hospital 4.0 (Hospital),

April 23-25, 2024, Hasselt, Belgium

Interoperability Architecture proposal for Adaptive Business Intelligence Systems in Healthcare Environments

João Guedes[a], Júlio Duarte[a,*], Maria Manuel[b], César Quintas[b], João Cunha[a], Tiago Guimarães[a], Manuel Filipe Santos[a]

[a]*ALGORITMI/LASI Researsh Center, University of Minho, Guimarães, Portugal*
[b]*Centro Hospitalar Universitário de Santo António, Porto, Portugal*

Abstract

The integration of systems for adaptive business intelligence (ABI) in the healthcare industry has the potential to revolutionize and reform the way organizations approach data analysis and decision-making. By providing real-time, actionable insights and enabling organizations to continuously adapt and evolve, ABI has the potential to drive better outcomes, reduce costs and improve the overall quality of patient care.
This article proposes an interoperability architecture that allows the seamless use and integration of HL7 messages and the information they contain as an interface with these systems. The proposed architecture follows a microservices paradigm with an emphasis on its modularity and distribution capacity and is part of an effort to achieve interoperability and ease of integration with ABI systems, while never neglecting key characteristics such as speed, security and the best practices in software development and architecture.

Keywords: ABI; Interoperability; HL7; Healthcare; Microservice

* Corresponding author. Tel.
E-mail address: jduarte@di.uminho.pt

10.1016/j.procs.2024.06.113

1. Adaptive Business Intelligence Systems and their potential application in healthcare

Adaptive Business Intelligence (ABI) systems are a new approach to Business Intelligence (BI), making use of Machine Learning and Artificial Intelligence systems and models to provide real-time information and insights into organizational data and aid in decision-making [1]. They are more agile than traditional BI systems and can automatically adjust and adapt to changes in data, business processes and the environment. This makes ABI systems very useful and suitable when applied in dynamic and unpredictable environments [1] [2].

ABI systems aim to provide organizations with a comprehensive and dynamic approach to data analysis, decision-making and continuous learning. They are made up of several interrelated components, including data collection and management, advanced analytics, modeling techniques and decision-making tools.

Over the years, organizations have collected astounding amounts of data in the hope that it could include useful information. Although the more data they have, the harder it becomes to pinpoint and extract useful information and insights about their business and environment. The true value of this stored unorganized data lies in the organization's capability for analysis. The healthcare sector is no different to the extent that there is a huge untapped potential for data analytics and insight extraction from stored data [2].

ABI systems for healthcare applications would be capable of extracting a lot of useful information and transforming data into information and insights; Forecasting models based on Data Mining Results; Run Optimization Models in full integration; Visualization of acquired knowledge through the ABI architecture [3].

Adaptive Business Intelligence systems are the result for when we combine, in the same system, predictive models that will try and predict the future based on historical data and optimization models that will try to fine tune the response of these predictive models as well as run as standalone models. In this way, ABI systems will always be able, in some capacity, to answer what will happen in the future and what is the best choice at the moment.

2. Interoperability System Architecture

The proposed system's focus is to create an environment and integration where interoperability resources (healthcare historical and daily data) can be used as input to run machine learning models in an ABI system.

The Health Level Seven (HL7) pattern was chosen as the central focus of the system due to its comprehensive complexity, flexibility and widespread adoption and support which will, practically guarantee compatibility with any healthcare system [4] [5] [6].

The proposed architecture follows the microservice approach and is divided into three separate and distinctive layers: **Technology**, **Services** and **Presentation**. Each of these layers contains a collection of loosely coupled services, virtualized into containers, where each one is self-contained and implements a single business capability. This kind of architecture will allow the interoperability system to have increased scalability and flexibility, improved maintainability, and testability as the different parts of the system become independent and easier to handle [7].

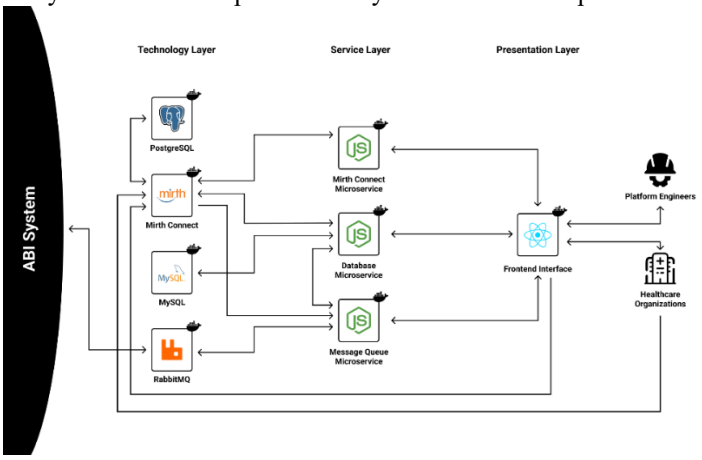

Fig. 1. Interoperability Architecture Representation

As it's represented in Figure 2, the proposed architecture has two end users in mind: **Healthcare Organizations** (clients), who would be using the system to run the many Machine Learning Models contained in the ABI system through HL7 resources, and **Platform Engineers** (System Administrators), who would be responsible for keeping the system running and maintaining all the rules. Every component of this architecture is an application, technology or service that was virtualized into a docker container for easy deployment and management [8], making it a multi-component system. This way the whole system can be installed/used through a single docker-compose file or any other docker container management solution [9]. All containers can be divided into different layers creating the **Technology, Service** and **Presentation** layers.

2.1. Technology Layer

This layer uses existing technologies to support the system. It's designed for both distributed and local systems, with optional technology containers that are only needed if not already present in the environment. In this layer there are four containers:

- Mirth Connect & PosgreSQL: Mirth Connect is an interoperability management engine that can handle HL7 resources natively [10] [11] and will be used to parse all incoming HL7 messages. It needs a separate Database to run properly, this database could be Derby, MySQL, PostgreSQL, Oracle or SQLServer;
- MySQL: MySQL was the Database of choice for the system because of its relational and robust nature as well as its high degree of integrability. This database will hold all the business information for the interoperability system;
- RabbitMQ: Message Queueing (MQ) was the method of choice for communicating with the ABI Systems (sending requests to run and receiving their responses) given that running machine learning models is a very volatile, and sometimes slow, task and MQ's asynchronous nature will make sure the requests do not overlap and all will get processed eventually and in a consistent fashion. Adding to this MQ also abstracts systems in communication of each other and this makes any kind of scaling very easy and flexible where the Interoperability system would put a request in queue and multiple ABI systems, services or applications would instantly be triggered.

2.2. Services Layer

This layer is composed of the microservices developed to interact with the technology layer and contain all the business logic. These microservices serve as interfaces for Mirth, MySQL and RabbitMQ and also make up the backend for the entire system taking the form of Web APIs that expose the necessary information through properly configured and secure endpoints. All these services were created using Node.js and Express.js. For security and access control, all sensitive routes are protected by a JWT Authentication Middleware and every sensitive variable/information (like credentials, secrets and IPs) is configurable through Environment variables. This layer is made up of three services [12] [13] [14]:

- Mirth Connect Microservice: This is the only service with access to the Mirth Connect credentials (through environment variables) and is responsible for all interactions that happen with it. It uses the built in Mirth Connect developer API which makes it possible to interact with Mirth Connect programmatically.
- Database Microservice: This service has access to the database's credentials through environment variables and is responsible for all interactions with the database for the system. For now, only MySQL is supported. This service utilizes sequelize [15], a library for secure interaction and querying of Databases in Node.js. This is arguably the most important service in the system because it contains all the logic and data to perform its' actions.
- Message Queue Microservice: This service has access to the credentials for the Message Queue technology, as it's responsible for all interactions with it. It, simultaneously, acts as a data publisher for the MQ but also as a listener, as all new incoming messages go through it. This is the way that the interoperability system communicates with its' integrations.

2.3. Presentation Layer

This layer is composed of all the components responsible for interacting with the final user being it clients or engineers and administrators. As of now, it consists of only one component, a single Frontend Application developed with React as the frontend framework. There are several advantages to using React, including its high performance and speed due to its smart virtual DOM, flexibility, scalability, large community, and ease of development[16].

3. Use Cases

There are hundreds of operations and use cases in the built system, for the context and scope of this article I'll simplify it to the two most important use cases that make up the heart of the system, sending and receiving HL7 resources.

3.1. Sending HL7 messages use case

There are two main ways to send a new request to the system, via the front-end application or via the exposed Mirth Channel (HTTP Web API). The only difference between these two variants is who is communicating with the Mirth Connect Channel, in one it's the actual client with their HTTP client and in the other it's the front-end application. Both, practically, start when the Mirth Connect Runner Channel is called. Succinctly, the flow for this use case is as follows:

- The Mirth Connect Channel will call the Database Microservice to get all HL7 mappings for the request model's features;
- The Mirth Connect Channel will, for every message in the batch, try to match the mappings to the provided messages and will start storing all matched information and in the end verify if all features were matched;
- If all features are matched and all necessary information is present and stored the Mirth Channel will send a request to the Database Microservice to check if all information is valid with the stored regex validations;
- If the response is valid that means all matched information is valid the Mirth Channel will send a request to the Database microservice to fetch the preprocessors and run them, in a sandbox environment to guarantee the safety of the system and to ensure only the data transformation happens, for every feature's data. This preprocessed data will, then, be sent back to the Mirth Channel to be stored in memory;
- After receiving the preprocessed data, the Mirth Channel will send a request to both Database Microservice and Message Queue Microservice to both store the actual data from the message and to forward the processed data to the Message Queue service to be consumed by any ABI system;
- The Mirth Channel will then send a response back to the client/front-end application with a success or failure message based on the previous checks.

3.2. Receiving model response from ABI system use case

After a successful request the data for running a Machine Learning model is published to the RabbitMQ requests queue and eventually it will be picked up by the ABI System which will run it in the desired model. After running the ABI will issue a response back to RabbitMQ responses queue, this response will be picked up by the Message Queue service and propagated to the Database which will be made available to the client through the graphical interface.

4. Solution Tests

To test the effectiveness and robustness of the solution **three healthcare related machine learning models** were implemented in the system and tested with real HL7 messages from the Healthcare Organization. In total there were **20 HL7 messages** none of which broke the system in any way and, to summarize, every test ended in success. The next section goes into more detail about the performed tests.

4.1. Message Completion

In this test it was assessed if the system is robust enough to run the requested models with the provided messages whether they have, or not, all the necessary information. To that end a formula to calculate the completion percentage of a group of messages was constructed as a testing metric.

$$TP_n = \frac{\sum_{i=1}^{n} P_i}{n} \times 100 \qquad P_i = \frac{CompleteFields_i}{TotalFields_i}$$

This test revealed that not all messages are suited to run all models, but that is where altering/customizing the messages/completing them, or the custom mapping mechanism comes into play and really shows how this issue can be tackled. As Figure 3 indicated, the test messages, if left as is, are very inconsistent in whether they will or will not work with the models, but when the messages are created and constructed with the mapping requirements in mind they all work or even creating custom mappings for different clients to fit their needs.

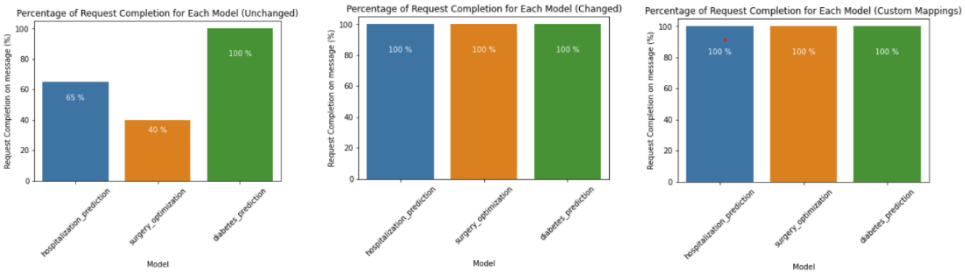

Fig. 2. Test Results of Messages As Is, Modified and Custom Mapped

4.2. Erroneous Messages

This test was conducted to test the robustness of the system when facing malformed or erroneous HL7 messages whether they were malformed HL7 messages with defective/wrong syntax or invalid data encoded into the message. There is an entire portion of the system dedicated to the validation of the input data. This portion will validate the data types, structure, and integrity according to user defined Regular Expressions based validations. If the system detects that the input data does not match the established validator, it will return an error and notify the user. As for the tests, all messages were tested with defective syntax and with erroneous data, the system was able to detect and handle all of the defects in the messages successfully, as is indicated inFig. 4.

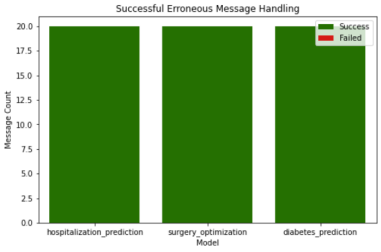

Fig. 3. Test Results of Erroneous Message Handling

5. Conclusion and Final Thoughts

This paper proposes an Interoperability system architecture for Adapative Business Intelligence systems that would allow for HL7 messages to be used as its very input. It was conceived with the best practices of software development in mind, guaranteeing robustness and stability in all use cases. It was developed using containerization for its modularity and simplicity of development, making all aspects of the system highly manageable and easy to

understand and maintain. The system employs Mirth Connect to link its various microservices. It receives HL7 messages, extracts data according to specific model feature mappings, and validates this data using pre-established validations based on regular expressions. The information is then preprocessed through custom scripts created by the final user. The processed data is subsequently forwarded to a RabbitMQ Queue, where it is executed by the ABI system, specifically targeting the model requested alongside the extracted data.
The system was built, developed, and then evaluated using a set of real-world messages. The results showed that it was able to successfully extract, validate, preprocess, and propagate the correct information in all the testing cases never breaking or returning erroneous information.
The proposed system was also devised to be very flexible and scalable, allowing it to easily adapt to new requirements and changes in the environment thanks to the mapping, validation and preprocessing solutions.
In the future, this system would benefit from supporting other Healthcare communication protocols such as FHIR or CDA, new interoperability technology integration and utility, thorough security assessments and improvements, data flow optimizations and extended database and message queue technology support.
The interoperability system can enhance ABI systems' implementation and adoption by Healthcare Organizations. It eliminates complex integration and enables the use and generation of insights from historical and daily data through HL7 resources.

Acknowledgements

This research was funded by Fundação para a Ciência e Tecnologia, within the Project Scope: UIDB/00319/2020.

References

[1] Michalewicz, Z., Schmidt, M., Michalewicz, M. and Chiriac, C. (2007). Adaptive Business Intelligence. 10.1007/978-3-540-32929-9.
[2] Lopes, J., Guimarães, T., & Santos, M. F. (2020). Adaptive business intelligence: A new architectural approach. Procedia Computer Science, 177, 540–545. https://doi.org/10.1016/j.procs.2020.10.075.
[3] Sousa, R., Miranda, R., Veiga, A., Alves, C., Lori, N. and Machado, J. "Software Tools for Conducting Real-Time Information Processing and Visualization in Industry: An Up-to-Date Review", *Applied Sciences*, Volume 11(11), MDPI, 2021.
[4] AlQudah, A. A., Al-Emran, M., & Shaalan, K. (2021). Medical data integration using HL7 standards for patient's early identification. PLoS ONE, 16(12 December). https://doi.org/10.1371/journal.pone.0262067.
[5] Miranda, M., Salazar, M., Portela, F., Santos, M., Abelha, A., Neves, J., & Machado, J. (2012). Multi-agent Systems for HL7 Interoperability Services. Procedia Technology, 5, 725–733. https://doi.org/10.1016/j.protcy.2012.09.080.
[6] Noumeir, R. (2019). Active Learning of the HL7 Medical Standard. Journal of Digital Imaging, 32(3), 354–361. https://doi.org/10.1007/s10278-018-0134-3.
[7] Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. Journal of Systems and Software, 150, 77–97. https://doi.org/10.1016/j.jss.2019.01.001.
[8] Potdar, A. M., Narayan, D. G., Kengond, S., & Mulla, M. M. (2020). Performance Evaluation of Docker Container and Virtual Machine. Procedia Computer Science, 171, 1419–1428. https://doi.org/10.1016/j.procs.2020.04.152.
[9] Hasan Ibrahim, M., Sayagh, M., & Hassan, A. E. (2021). A study of how Docker Compose is used to compose multi-component systems. Empirical Software Engineering. 26. 10.1007/s10664-021-10025-1.
[10] Lin, J., Ranslam, K., Shi, F., Figurski, M., & Liu, Z. (2019). Data migration from operating EMRs to OpenEMR with mirth connect. Studies in Health Technology and Informatics, 257, 288–292. https://doi.org/10.3233/978-1-61499-951-5-288.
[11] Rodriguez, J. C. C., Stäubert, S., & Löbe, M. (2017). Automated import of clinical data from HL7 messages into open clinica and tran SMART using mirth connect. Studies in Health Technology and Informatics, 228, 317–321. https://doi.org/10.3233/978-1-61499-678-1-317.
[12] Doglio, F. (2018). REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development, Second Edition. In REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development, Second Edition. Apress Media LLC. https://doi.org/10.1007/978-1-4842-3715-1.
[13] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020.
[14] Montenegro, L., Peixoto, H. and Machado, J. "Evaluation of Transfer Learning to improve Arrhythmia Classification for a small ECG Database", IBERAMIA 2022, LNCS Volume 13788, Springer, 2022.
[15] Barsoti, N., & Gibertoni, D. (2020). IMPACTO QUE O SEQUELIZE TRAZ PARA O DESENVOLVIMENTO DE UMA API CONSTRUÍDA EM NODE.JS COM EXPRESS.JS. Revista Interface Tecnológica, 17(2), 231–243. https://doi.org/10.31510/infa.v17i2.964.
[16] Rawat, P., & Mahajan, A. N. (2020). ReactJS: A Modern Web Development Framework. In International Journal of Innovative Science and Research Technology (Vol. 5, Issue 11). www.ijisrt.com.