Reusability of Interfaces in Healthcare EAI Environments

Severin Linecker[1,2] and Wolfram Wöß[1]
[1]*Johannes Kepler University Linz, Austria*
[2]*Vinzenz Gruppe, Linz, Austria*

Keywords: Enterprise Application Integration, EAI, Middleware, Message-oriented Middleware, MOM, HL7, System Integration, Healthcare.

Abstract: Enterprise Application Integration (EAI) and HL7 (Health Level Seven) messaging are well established technologies in healthcare environments. Due to the widely adoption of HL7 messaging, especially the version 2, in the healthcare domain and its flexibility, many vendor specific implementations exist. To integrate these systems, messages have to be adapted to the vendor specific requirements, even if the functionality is nearly the same. This leads to an increasing number of special interfaces and decreased maintainability. This paper shows a generic architecture for reusable interfaces for HL7 messaging by considering reusability at data level and interface level and the results when applied to a real production EAI environment of an austrian healthcare provider.

1 INTRODUCTION

Exchanging clinical data between multiple heterogeneous medical information systems is very common in healthcare environments. The Hospital Information System (HIS) and other special (sub)systems, such as the Radiological Information System (RIS) need to be integrated for digital clinical workflows. Therefore Enterprise Application Integration (EAI) and HL7 (Health Level Seven) messaging are well established technologies in healthcare environments.

Due to the message oriented nature of HL7, Message-oriented Middleware (MOM) is a common paradigm for implementing EAI in healthcare environments (Bezerra et al., 2015). It allows systems to communicate with each other by sending and receiving messages using interfaces directly connected to the middleware, which is then responsible for routing these messages to their correct destinations. This helps to reduce the total number of interfaces needed to connect n systems from $(n*(n-1))/2$, when using point-to-point interfaces, to n. Especially for complex and big healthcare environments, which often consist of 50 or more connected systems (like in the case of the Vinzenz Gruppe, an association of seven religious-order hospitals and other healthcare facilities in Austria), this is a necessity. HL7 is a messaging standard specifically developed for exchanging data between information systems in healthcare environments. Version 2 (HL7 V2.x) of the messaging standard was first released in 1987 and, according to (HL7 International, 2020), is one of the most widely used standard for healthcare information exchange. The HL7 V2.x standard (ISO, 2009) defines message types and their (real world) trigger events for clinical, financial and administrative data exchange. Messages have a data type and a trigger event, which together define a specific sequence of segments and segment groups. For example, the `ADT` (Admission, Discharge and Transfer) message type and the trigger event `A02` is used for transmitting patient administration information about a patient transfer. The following listing shows the message type definition of an HL7 `ADT^A02` message:

```
MSH          Message Header
[{ SFT }]    Software Segment
EVN          Event Type
PID          Patient Identification
[ PD1 ]      Additional Demographics
[{ ROL }]    Role
PV1          Patient Visit
[ PV2 ]      Patient Visit - Additional Info.
[{ ROL }]    Role
[{ DB1 }]    Disability Information
[{ OBX }]    Observation/Result
[ PDA ]      Patient Death and Autopsy
```

Segments are a logical grouping of data fields. They may be required, optional or repetitive within a message type. Two or more segments may be

Linecker, S. and Wöß, W.
Reusability of Interfaces in Healthcare EAI Environments.
DOI: 10.5220/0010242004170423
In *Proceedings of the 14th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2021) - Volume 5: HEALTHINF*, pages 417-423
ISBN: 978-989-758-490-9

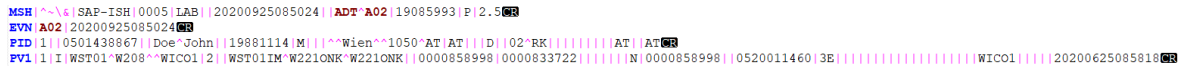

Figure 1: Example HL7 V2.5 `ADT^A02` message.

grouped together as a logical unit. Each segment has a unique name called Segment ID with three upper case characters (e.g., `MSH`, `PID`, `PV1`). Fields are character strings within a segment. They have an ordinal position within the segment for reference (e.g., `PID.3`), have a data type, can be required or optional and may be repetitive. Depending on its data type, a field may consist of components, which in turn may contain subcomponents. HL7 V2.x messages are typically plain text messages with certain special characters used as delimiter. They can be negotiated between applications, but typically the HL7 recommendation is used. Figure 1 shows an `ADT^A02` message with the HL7 V2.x typical delimiters `|` for fields, `^` for components, `&` for subcomponents, `~` for repetitive fields and \ as escape character. Segments are always terminated by a carriage return.

No matter how detailed the specification is, HL7 V2.x is a standard for syntactic but not for semantic interoperability. The context of a field is defined in the standard, but leaves the implementers the flexibility to use a free text or a coded value from a standard terminology. These details have to be negotiated between the issuer and the recipient of a message to ensure correct interpretation. This step is not done automatically, but usually in advance by the interface developers of the corresponding systems.

In hospital environments the HIS is a central point for data exchange. The connected systems send their data to the HIS, but also require its data. Especially patient administration data is required by most of the systems. These are distributed to them with HL7 `ADT` messages using middleware technology. The problem is that due to the flexibility of the HL7 standard, the messages often have to be adapted to meet the requirements of the receiving system. This leads to an increased number of special interfaces and decreased maintainability. In order to save time and resources during the development of integration projects, it is important to have reusable interfaces available that allow these adaptions using configuration. To meet the objective above, this work introduces a generic architecture for creating reusable interfaces, by considering reusability 1) at data level by splitting up message flow in multiple interfaces and 2) at interface level by using twelve reusable components.

The remainder of the paper is organized as follows: Section 2 contains related work. In Section 3 the two levels of the architecture are described in detail. In Section 4 the evaluation results of applying the proposed architecture to a real production healthcare EAI environment are shown. The paper finishes with Section 5 with conclusion and future work.

2 RELATED WORK

The authors of (Bezerra et al., 2020) show a middleware architecture consisting of a cloud service and local clients. Reusability was taken into account by separating the business rules from the persistence layer on the client side, and on the server side by providing the middleware functionalities via a central cloud service. Reusability is also considered in the work of (Li-Fan Ko et al., 2006), where a SOA based middleware framework for healthcare information systems was proposed. Another SOA based EAI approach (non healthcare) is done by (Jun Gui and Hebiao Yang, 2010), where service components can be reused for existing or newly developed components.

Even though there are some more studies about HL7, healthcare information exchange, EAI and middleware architecture in the healthcare domain like (Alenazi and Alhamed, 2015), (Wadhwa et al., 2015), (Liu and Huang, 2012), (Lu et al., 2010), (Bortis, 2008) or (Vargas and Pradeep Ray, 2003), their main objective is not reusability. Their concern is the integration problem itself and not reusing integration components for further integration needs.

The contribution of this work is a generic architecture for HL7 based messaging, used for implementing highly reusable interfaces and components in (existing) healthcare EAI environments.

3 ARCHITECTURE

In this section, an architecture for interfaces is presented, where reusability is considered in all parts of an EAI environment. This includes the general design of message flows between systems within the EAI environment (data level), which is shown in Section 3.1, and a component based architecture for implementing interfaces (interface level), shown in Section 3.2.

3.1 Messaging

Instead of creating one single interface for connecting source and destination system(s), message paths

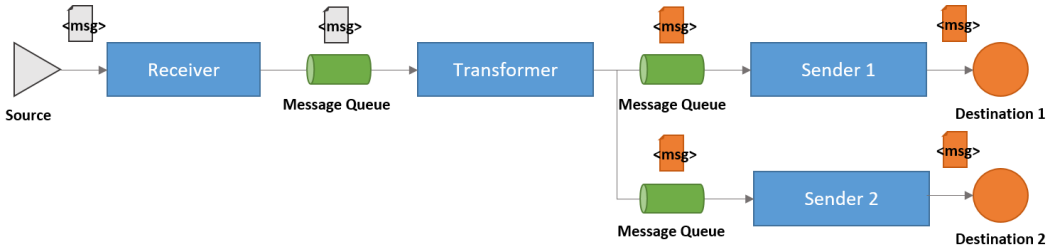

Figure 2: Asynchronous message flow between multiple systems.

should be composed out of multiple interfaces interconnected with message queues. Figure 2 shows an asynchronous message path from a source system to two destination systems split up into three steps, each with different responsibilities. This architecture allows branching off messages after each step, reusing the output of one interface as an input for another interface. The following sections describe the functionality of each interface in more detail.

3.1.1 Receiver

The Receiver interface gets the message from a source system via a communication channel (File, HTTP, MLLP, SOAP, REST, ...). It checks the syntactical correctness of the received message and generates an appropriate response if required by the used communication protocol. In HL7 messaging the Minimal Lower Layer Protocol (MLLP) is widely used for sending and receiving HL7 messages via network sockets. After checking syntactical correctness, the business keys of the message are extracted and stored together with the raw message in a message backup/journal. This is needed for searching and resending messages manually in case of an error. Finally, the message is passed to one or many message queues or gets filtered due to specified filter criteria. Afterwards, processing of the receiver interface ends.

Errors during message processing are reported to the administrator, who is responsible for monitoring the EAI environment. This applies to all other interfaces as well and is not mentioned anymore in the following sections.

3.1.2 Transformer

First, the Transformer interface reads the message from a message queue. Afterwards, it transforms the message as needed. This can be format conversion, like transforming an XML message or a proprietary HIS message format to an HL7 V2.x message, or performing changes to a HL7 message. Finally, the transformed message is passed to one or many message queues or gets filtered. Processing of the Transformer interface ends afterwards.

The Transformer interface is not needed in all message streams between source and destination system and can be omitted in cases where no message transformation is needed at all. But even if there is no transformation needed, there are scenarios that require special routing and filtering capabilities between Receiver and Sender interfaces. In this case the transformation part is skipped and only the routing and filtering part is done. These interfaces are called Router or Forwarder interfaces.

3.1.3 Sender

The Sender interface reads a message from a message queue. It checks the syntactical correctness and the presence of all needed data for the destination system. Like the Receiver interface, it also extracts business keys from the message and stores them with the raw message in a backup/journal. Finally, the message is transmitted to the destination system using a communication channel (e.g., File, MLLP, SOAP, REST, DB, SAP RFC). Restricting the number of outbound communication channels to one has reliability reasons. If Sender 1 in Figure 2 would also send to Destination 2, and Destination 1 is not reachable, messages could not be delivered to Destination 2 as well.

3.2 Interface Components

Reusable interfaces are built to reduce the amount of effort and time needed to build integration solutions. The main idea is to build an abstraction layer on top of an existing EAI platform to abstract limitations regarding reusability and interface architecture in general, and to provide a generic API for interface development. The goal is that code, once written, can be reused multiple times, saving a vast amount of time in integration projects and in case of migrating to new platforms.

Reusability is achieved by splitting up an interface implementation into different components, each with its own responsibility within the whole integration task. Figure 3 shows twelve components used for interface implementation. The provided structure and

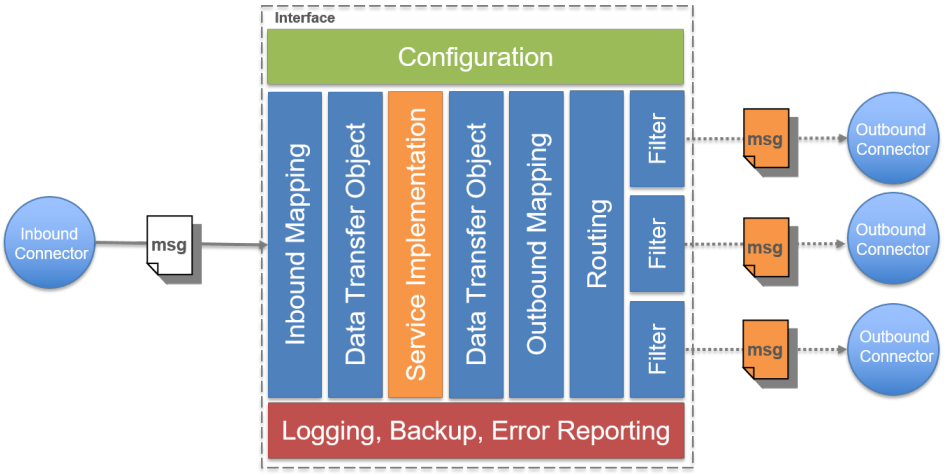

Figure 3: Components of an interface.

components can be implemented on top of an existing EAI platform, or for standalone integration solutions.

The following twelve components are used to build (reusable) interfaces:

- **Inbound Connector:**
  The Inbound Connector component is used for reading raw messages (e.g., text messages) from a communication channel (File, Message Queue, Socket, HTTP, ...).
- **Data Transfer Object:**
  A Data Transfer Object (DTO) defines a data structure, which is used by the Service Implementation during processing. It is an abstract representation of the data being used by an interface. DTOs form the basis for generic reusable interfaces by providing access to the required data in a message structure independent way.
- **Inbound Mapping:**
  The Inbound Mapping transforms the incoming raw message to a DTO structure. Therefore it uses appropriate parsers (e.g., HL7 parser) to verify the syntactic correctness of the incoming message and to access its fields for extraction.

  After parsing and DTO field extraction, the data has to be verified. This includes checking the presence of all required fields and their semantic correctness. After the successful completion of all Inbound Mapping steps, the result is a DTO message containing all fields necessary for the Service Implementation component to perform its task. Errors during parsing or field validation get logged by the Logging component and/or reported using the Error Reporting component.
- **Service Implementation:**
  The Service Implementation component is responsible for performing the business logic of the interface. This is the main part of any interface implementation. It only operates on DTOs and is therefore independent of the original and/or target message structure. Accessing other data, which is not present in the DTO, requires the usage of Data Access Objects (DAO).
- **Outbound Mapping:**
  The Outbound Mapping component performs the inverse operation to an Inbound Mapping. It takes a DTO and transforms it to a target message structure (e.g., HL7 V2.x message). If necessary, character set transformation and proper escaping of certain special characters is done here.
- **Routing:**
  Selecting the correct destination Outbound Connector for a given message is done by the Routing component. Static and dynamic routing is supported and is achieved using configuration or Filter components.
- **Filter:**
  A Filter component produces a boolean output for a given arbitrary input. Filters can be conjuncted with the logical operations `and`, `or` and `not`.
- **Outbound Connector:**
  Outbound Connectors are used for sending raw message data to an outbound communication channel (File, Message Queue, Socket, HTTP, ...).
- **Configuration:**
  Each component has to provide useful configuration options, but always has a runnable default configuration suitable for most integration scenarios. The principle convention over configuration applies here. Each interface has its own configuration possibility, containing all configuration options set for all components.

- **Logging:**
  The Logging component is used by all other components. For traceability of message flows at least the incoming and the outgoing message(s) should be logged.
- **Backup:**
  The Backup component is used on messages entering and leaving the EAI environment. The raw data and business keys of the message are stored in a database to enable searching and resending of messages in case of errors during processing.
- **Error Reporting:**
  When an error occurs, the Error Reporting component is used to send notifications to responsible addressees. To distinguish between different error conditions, hierarchical error codes are used.

Not every component has to be used in a specific interface implementation. They are to be seen as building blocks that enable or facilitate the development of interfaces. There are three different approaches to achieve reusability at interface level:

1. Implementation of multiple generic components that serve as building blocks for interfaces. The interfaces themselves cannot be reused, but their individual components can. For example, a generic Filter component can be implemented which checks the content of a field. Both the specific field (e.g., `PID.3` from an HL7 message) and the check itself (e.g., a REGEX) can be provided as a configuration option. This approach is most suited for environments having a (manageable) range of functionality that has to be composed in many different ways and there is less or no need of reusing an interface as a whole.
2. Implementation of interfaces using type specific components. Here the interfaces are reusable themselves, but not their individual components. Therefore the components have to be reimplemented for each type of interface. This approach is best suited for (smaller) environments where interfaces can be grouped together based on their functionality. For example, having only interfaces for patient administrative data using HL7 `ADT` messages. This approach also suits well for simple ad-hoc integration solutions, which may not need an implementation for all different component types.
3. A hybrid approach, where the implementation of interfaces uses both, generic and specific components. This approach suits well for environments, having interfaces of certain types (patient administration, observations results, MLLP communication, ...) and also special interfaces which do share some functionality. Furthermore some components are available in a generic reusable way (Logging, Backup, Error Reporting, Routing, Filter, ...).

4 EVALUATION

This architecture was implemented and deployed in the EAI production environment of the Vinzenz Gruppe. At the end of 2014 the production EAI environment consisted of 342 interfaces deployed to three production servers. Each of these interfaces was implemented using its own Java class, thus class reusage was not present at all, even many of the existing interfaces did similar tasks. Code reusage was primarily based on copy and paste with subsequent manual adaptation. As a result many interfaces had a common basis, but there were numerous extensions and variants concerning for example, the message structure, the use of free text fields, additional fields and data mappings.

With the beginning of 2015 an analysis was initiated to find interface categories sharing common functionality. The hybrid approach for interface reusability was then used to implement a configurable standard implementation for each interface category. The four most important ones were: a MLLP Sender interface [A], a MLLP Receiver interface [B], an interface for routing and forwarding of arbitrary messages [C], and a standard HL7 ADT Transformer interface [D]. Furthermore, configurable generic components were implemented for functionalities that were required by many interfaces. These were Logging, Backup, Error Reporting, Routing, Filter, Inbound- and Outbound Connector components.

For new interfaces the standard interfaces were used where appropriate, and the rollout was done by configuring instances of them accordingly. Furthermore old legacy interfaces were replaced with the equivalent standard implementation whenever possible, or have been reimplemented with the proposed architecture. This led to an increasing number of reusable components and interface classes.

The current EAI production environment consists of 534 interfaces, deployed to seven production servers. 86 interfaces are legacy interfaces which do not conform to the architecture presented in this paper. Table 1 shows the number of interface classes and the number of instances of each class running in the production environment.

The pie chart of Figure 4 shows that the top four interface classes which have a reusage count $N >= 25$

Table 1: Reusage statistics of different interface classes.

N.o. interface classes	Instances each	Sum
1 [A]	70	70
1 [B]	46	46
1 [C]	45	45
1 [D]	27	27
3	9	27
4	8	32
9	7	63
2	6	12
5	5	25
4	4	16
3	3	9
11	2	22
54	1	54
legacy	86	86

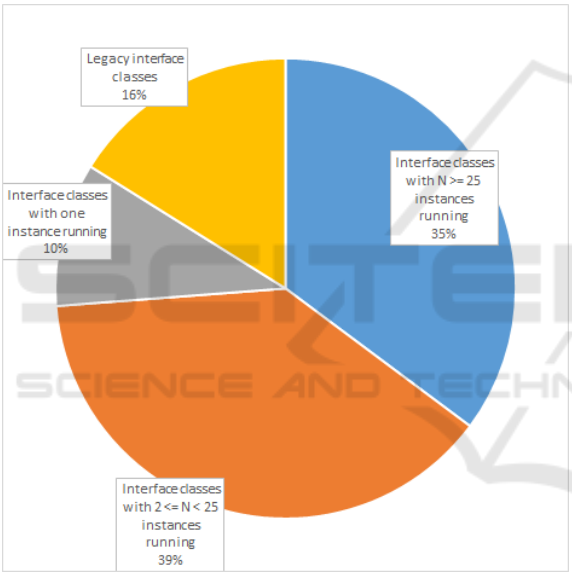

Figure 4: Pie chart of interface class reusage.

make up 35 % of all interfaces running. Adding the 39 % of all interface classes with a reusage count $2 <= N < 25$, 74 % of all running interface instances reuse an interface class. Using the proposed architecture we could increase interface class reusage from 0 % to 74 % which helped saving time and money on new integration projects.

5 CONCLUSION AND FUTURE WORK

In this paper an architecture for reusable interfaces using a multi step message flow design and reusable components for interface implementation was shown. The introduction of an abstraction layer between an EAI platform and the interface implementations eases the implementation of generic reusable interfaces. Code, that has been written once, can now easily be reused either as a component in a new interface implementation or as an entire interface. This helps to save resources when developing new integration projects.

The abstraction of the EAI platform is the first step to replace the existing system with a new one. It is planned to analyse the reusability capabilities of interfaces implemented with the architecture shown in this paper when migrating them to a new EAI platform.

REFERENCES

Alenazi, T. M. and Alhamed, A. A. (2015). A middleware to support hl7 standards for the integration between healthcare applications. In Balakrishnan, P., Srivatsava, J., Fu, W.-T., Harabagiu, S. M., and Wang, F., editors, *ICHI*, pages 509–512. IEEE Computer Society.

Bezerra, C., Araujo, A., Sacramento, B., Pereira, W., and Ferraz, F. (2015). Middleware for heterogeneous healthcare data exchange: a survey. In *ICSEA 2015 Tenth International Conference on Software Engineering Advances*, pages 409–414.

Bezerra, C. A. C., de Araújo, A. M. C., and Times, V. C. (2020). An hl7-based middleware for exchanging data and enabling interoperability in healthcare applications. In Latifi, S., editor, *17th International Conference on Information Technology–New Generations (ITNG 2020)*, pages 461–467, Cham. Springer International Publishing.

Bortis, G. (2008). Experiences with mirth: An open source health care integration engine. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, page 649–652, New York, NY, USA. Association for Computing Machinery.

HL7 International (2020). Hl7 version 2 product suite. https://www.hl7.org/implement/standards/product_brief.cfm?product_id=185. Last checked on Sep 09, 2020.

ISO (2009). ISO/HL7 27931:2009 Data exchange standards – health level seven version 2.5 – an application protocol for electronic data exchange in healthcare environments. http://www.iso.org/iso/catalogue_detail.htm?csnumber=44428.

Jun Gui and Hebiao Yang (2010). Realization of eai based on service-oriented architecture. In *2010 International Conference on Educational and Information Technology*, volume 2, pages V2–424–V2–428.

Li-Fan Ko, Jen-Chiun Lin, Chi-Huang Chen, Jie-Sheng Chang, Faipei Lai, Kai-Ping Hsu, Tzu-Hsiang Yang, Po-Hsun Cheng, Chia-Chang Wen, Jun-Lian Chen, and Siao-Lin Hsieh (2006). Hl7 middleware framework for healthcare information system. In *HEALTHCOM 2006 8th International Conference on e-Health*

Networking, Applications and Services, pages 152–156.

Liu, L. and Huang, Q. (2012). An extensible hl7 middleware for heterogeneous healthcare information exchange. In *2012 5th International Conference on BioMedical Engineering and Informatics*, pages 1045–1048.

Lu, X., Gu, Y., Yang, L., Jia, W., and Lei Wang (2010). Research and implementation of transmitting and interchanging medical information based on hl7. In *The 2nd International Conference on Information Science and Engineering*, pages 457–460.

Vargas, B. and Pradeep Ray (2003). Interoperability of hospital information systems: a case study. In *Proceedings 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (HealthCom)*, pages 79–85.

Wadhwa, R., Mehra, A., Singh, P., and Singh, M. (2015). A pub/sub based architecture to support public healthcare data exchange. In *2015 7th International Conf. on Communication Systems and Networks (COMSNETS)*, pages 1–6.