HIE

A standards-based distributed Healthcare Integration Engine

Instituto Superior Técnico

Ricardo Miguel Simões Soeiro
ricardo.soeiro@tagus.ist.utl.pt

Abstract

The main goal of this work is to propose a new integration system named HIE that is designed to satisfy the specific needs of the healthcare market. This system should allow the establishment of communication channels between heterogeneous healthcare systems. To achieve this goal, the system must support the most used interoperability standards in this domain like HL7 or DICOM, along with other general-purpose ones like XML, Web Services or Databases. The system should be extensible to support the future addition of other standards and it should also be configurable through an accessible interface that doesn't require a deep knowledge of the used standards.

The purpose of this article is to provide a high level overview of the key concepts used in the system, without going into implementation details.

This system is being developed at the software company Infortucano, specialized in healthcare software solutions.

Keywords: interoperability, standards, integration engine, HL7, DICOM, XML

Introduction

Interoperability between heterogeneous systems is one of the oldest issues in IT and is still considered a problem today. Many organizations can benefit from the sharing of information between its systems, with better accessibility to their data and more agility in their business processes. In the healthcare domain, communication between information systems improves the quality of the provided health services, reducing the need for manual (re)introduction of data and improving the quality of medical decisions with better access to vital information. [1]

In the last decades there were significant breakthroughs with the creation of general-purpose interoperability standards like EDI or XML and domain-specific ones like HL7 or DICOM [2], for healthcare. These standards have made it easy to create point-to-point interfaces between information systems. However, the exponential growth [3] of these types of interfaces has raised some problems to the organizations, mostly related to their maintenance.

In order to avoid these issues, organizations started turning towards integration middleware, commonly known as integration engines. This type of software is specialized in the creation of interfaces using different communication standards, providing tools to manage them centrally. Integration engines have been gradually replacing point-to-point interfaces, although these are still very common today.

In the healthcare domain there are still many flaws in this type of products, from the absence of support for some of the most used standards, to the lack of flexibility provided for configuring interfaces and even some performance and scalability issues.

Architecture

The architecture of the HIE system is based on the following set of architectural patterns and principles:

- Enterprise Service Bus & SOA
- Layers
- Modular Design

Enterprise Service Bus & SOA

One of the main concepts behind the architecture of the system is the Enterprise Service Bus, an architectural pattern proposed by Dave Chappell. [4]

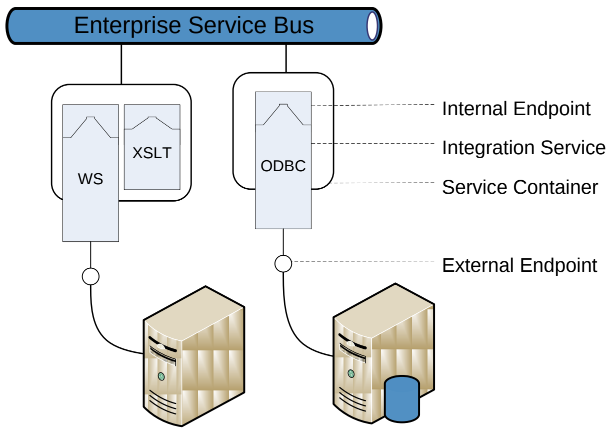


Figure 1 - ESB Architectural Pattern

The main features of an ESB are the following:

- **Message Bus** that can be provided by a regular MOM (Message-Oriented Middleware);
- **Integration Services** that can provide external endpoints for other systems to connect to, like Web Services or HL7 interfaces or other utility services like Transformations or CBR (Content-Based Routing);
- **Service Containers** where different integration services are deployed, provided with internal endpoints to exchange messages with other services or the parent container through the message bus.

Architectures based on this pattern allow the service containers to be deployed in different machines, providing a distributed system.

The ESB pattern is intimately related to **SOA** (Service-Oriented Architecture) principles. With an ESB we can create different instances of reusable integration services and then combine them to form of service orchestrations. An orchestration can then be made available for other systems by defining an external endpoint (like a Web Service) and linking it to the orchestration.

Layers

Another architectural pattern used in the conception of the system is the layers pattern [5], where the system is structured according to different layers of functionality that depend on the layers below.

The layers defined for this system are the following:

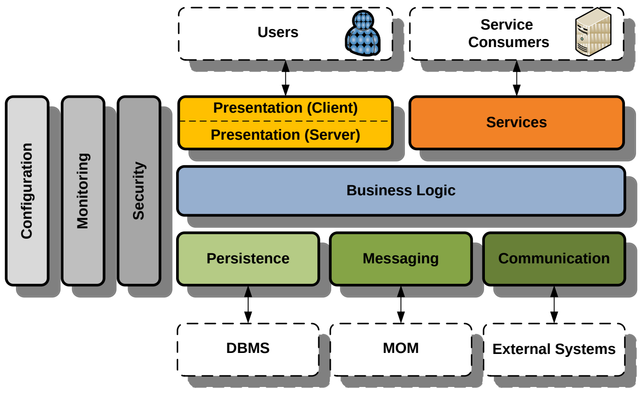


Figure 2 - Layers of the HIE system

- **Presentation** – responsible for the interface that allows the user to configure the different parts of the system. This layer is subdivided into client-side and server-side components. The client-side components are required for complex user interactions like visual mapping between message schemas.
- **Services** – this layer provides the external endpoints used by external systems to connect to this system;
- **Business Logic** – contains the core classes of the system like domain objects and utility classes;
- **Persistence** – used for storage and recovery of data using a database;
- **Messaging** – provides low level functions used to communicate with the MOM for sending and receiving messages;
- **Communication** – provides the lower level mechanisms required for establishing connections with external systems using different communication protocols (like DICOM or HL7);

This model also contains three different "zones" that represent important system-wide policies:

- **Security** – security-related policies that try to protect the system from a six categories of threats defined by the STRIDE acronym [6]:
    - **S**poofing;
    - **T**ampering;
    - **R**epudiation;
    - **I**nformation Disclosure;
    - **D**enial of Service;
    - **E**levation of privilege.
- **Monitoring** – a set of rules that defines how to monitor the different components of the system through four different levels of event logging:
    - Service Container Event Logs;
    - Integration Service Event Logs;
    - User Activity Event Logs;
    - Message Logs;
- **Configuration** – a set of principles that define which parts of the system should be configurable and which ones shouldn't.

Modular design

The last architectural pattern that is used by the HIE system consists of the separation of its parts into different components that can be developed independently.

The components of the system are the following:

- ***Service Container*** – responsible for hosting integration services, creating a different *thread* for each one. The container is also responsible for managing the life cycle of its services;
- ***Base Service*** – generic base from which every integration service inherits parts of its features, allowing them to focus only on providing their business value;
- ***Domain Classes*** – informational entities managed by the system;
- **Support Classes** – DAO classes, helper classes and other utilities;
- **HL7 Utilities** – HL7 support classes that allow us to parse, encode, send and receive HL7 messages;

- **DICOM Utilites** – DICOM support classes that allow us to parse, encode, send and receive DICOM messages;
- **XML Utilities** – XML utilities that allows us to parse, encode, query, (XPath) transform (XSLT) and validate (XSD) XML documents;
- **Communication Classes** – classes that implement lower level protocols like TCP/IP;
- **Persistence Classes** – classes that implement lower level communication with a DBMS (DataBase Management System);
- **Messaging Classes** – classes that implement lower level communication with the MOM that implements the message bus;
- **Integration Services** – classes that implement the abstract methods of the Base Service, providing new business value for the system;
- **Web Interface** – presentation-related classes that provide the user interface.

Implemented Integration Services

The HIE is mainly composed of service containers and integration services that are hosted by the containers. There are two kinds of services:

- **Interface services** – these provide external endpoints used by other systems to access the HIE;
- **Internal services** – these provide some utilities like message routing or transformation that are key elements in an orchestration;

Every service has an internal endpoint available to exchange messages with other services deployed in the system. The contents of the messages are always based on XML and must reference the XSD schema used to validate its contents. This ensures that only valid messages flow through the bus, allowing them to be routed and transformed by internal services. On the other hand, this means that every service that receives input from an outside source must convert that information to XML before sending it to another service, which means that automatic converters should be used for messages that use other encodings.

There are two types of interactions between services:

- **Request-only** – a message is sent and the service resumes its previous state;
- **Request-response** – a message is sent and the service blocks, waiting for a response message in its message queue. The request-response pattern is required for synchronous operations like Web Service interactions that need to send a response to the requester and that response is generated in another service;

In the current prototype there are ten different service types that represent the building blocks of the orchestrations. These services will now be described:

- **Transformation** – takes an input XML, transforms it into another XML document and validates the result before sending it to another service;
- **Content-Based Routing** – forwards a message to another service based on its contents, which is verified with XPath expressions;

- **HL7 Receiver** – receives HL7 messages and converts them to XML if their schema is recognized. This service can be configured to implement request-response interactions;
- **HL7 Transmitter** – sends an HL7 message to one or more external systems and can wait for a response, if configured accordingly;
- **DICOM Worklist SCP** – this service implements a DICOM Worklist SCP (Service Class Provider), allowing imaging modalities to query it for scheduled procedures steps;
- **DICOM MPPS SCP** – implements a Modality Performed Procedure Step SCP that can receive data related to exams performed by imaging modalities. The received information can be converted to XML and sent to other services;
- **DICOM *Query/Retrive* SCU** – this service can query remote Storage SCPs for images or other DICOM objects and ask these systems to send these objects;
- **Database** – this service can send SQL commands to databases. When configured to send queries, the system receives the data and converts it to the XML format following the schema of the data table from where the results are retrieved;
- **Web Service** – this service is configured to receive SOAP requests from other systems, making its WSDL contract available so that others can build client stubs from it in order to use the service;
- **Web Service Caller** – this service sends SOAP requests to external web services. The remote web service schema is extracted previously and used to configure the service automatically.

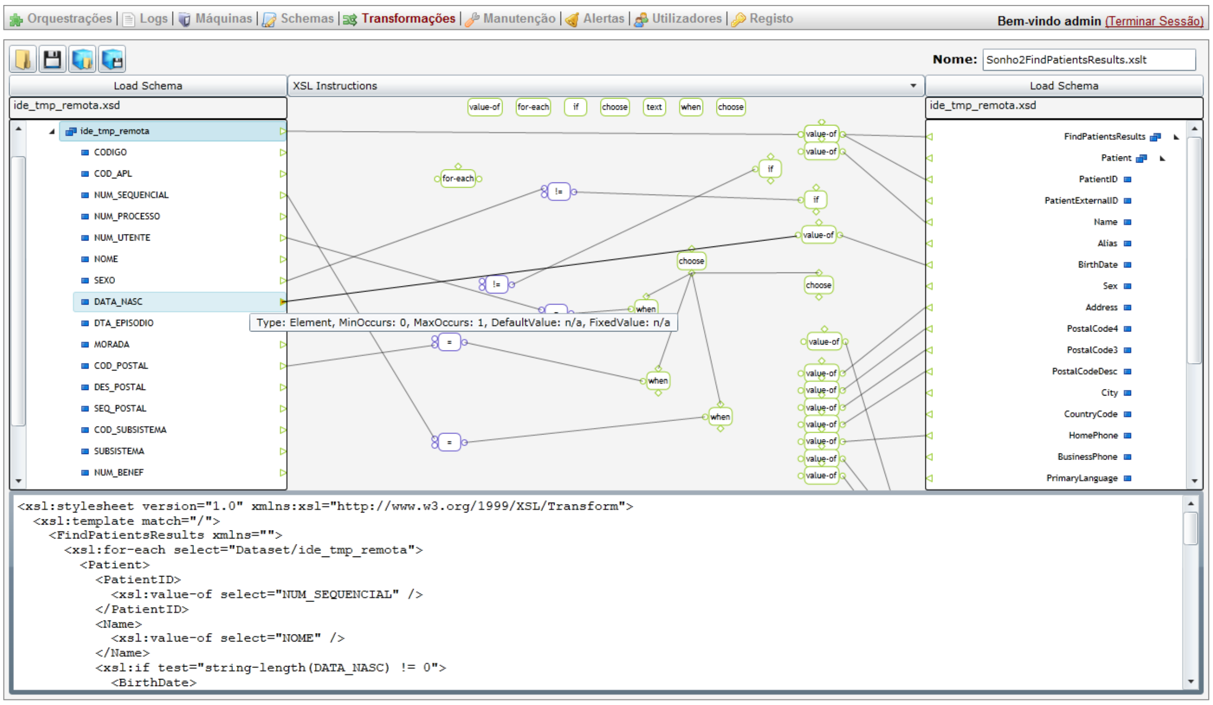
Web Interface

The Web interface allows authorized users to configure all the aspects of the system, review the logs and message queues and monitor the status of the integration services. The interface is divided into several components that are independent and require different access permissions from the users registered in the system.

The components of the interface are:

- **Orquestration management** – allows users to create, delete and modify orchestrations (by adding or removing service instances);
- **Service Configuration** – this component allows a user to configure the parameters of a service instance;
- **Schema management** – used to create, delete and modify XSD schemas, allowing the user to import schemas (from files, WSDLs or database tables);
- **Transformation management** – used to create, delete and modify transformations (XSLT-based). This component allows the user to create mappings visually with a drag-and-drop interface, using a client-side control;
- **User and permissions management** – component used to create, delete and modify users and domains;
- **Message Queue management** – component used to review the contents of a service's message queue, allowing the user to remove and re-inject messages;
- **Logs** – this is where the users can review the four different types of event logs related to containers, services, messages and user activity;

- **Device Configuration** – in this component the user is allowed to manage devices known by the system and the endpoints associated with each device;
- **Alert Configuration** – here the user can define alerts associated with certain events that occur in a service. The alerts are sent to the user by e-mail as they occur or when a certain time/quantity threshold is reached;
- **System Management Configuration** – this component contains the configurations related to the automatic archive and clean-up of logs and message history, for maintenance purposes



```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <FindPatientsResults xmlns="">
      <xsl:for-each select="Dataset/ide_tmp_remota">
        <Patient>
          <PatientID>
            <xsl:value-of select="NUM_SEQUENCIAL" />
          </PatientID>
          <Name>
            <xsl:value-of select="NOME" />
          </Name>
          <xsl:if test="string-length(DATA_NASC) != 0">
            <BirthDate>
```

Figure 3 - Screenshot of the Visual Mapper

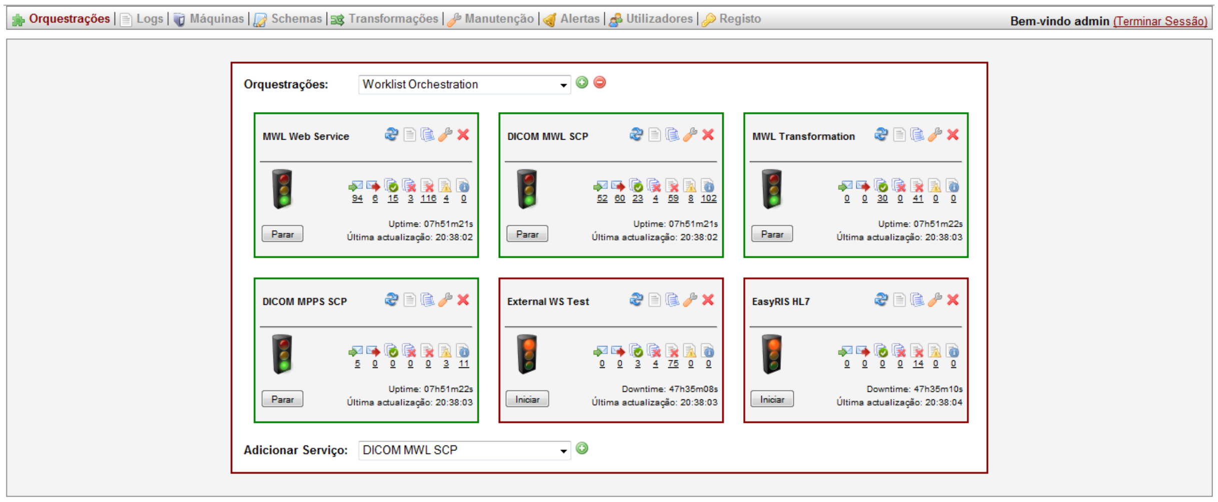

Orquestrações | Logs | Máquinas | Schemas | Transformações | Manutenção | Alertas | Utilizadores | Registo
Bem-vindo admin (Terminar Sessão)
Orquestrações: Worklist Orchestration
MWL Web Service
Parar
Uptime: 07h51m21s
Última actualização: 20:38:02
DICOM MWL SCP
Parar
Uptime: 07h51m21s
Última actualização: 20:38:02
MWL Transformation
Parar
Uptime: 07h51m22s
Última actualização: 20:38:03
DICOM MPPS SCP
Parar
Uptime: 07h51m22s
Última actualização: 20:38:03
External WS Test
Iniciar
Downtime: 47h35m08s
Última actualização: 20:38:03
EasyRIS HL7
Iniciar
Downtime: 47h35m10s
Última actualização: 20:38:04
Adicionar Serviço: DICOM MWL SCP

Figure 4 - Screenshot of the Orchestration Management

Technologies

There were some initial requirements from the stakeholders of the system, stating that the system should be developed using the **.NET Framework** from Microsoft.

With this requirement in mind, all the other technologies that were integrated into the system use this framework.

The **NUnit framework** was used for supporting the TDD (Test-Driven Development) of the domain model and its support classes

The DBMS used is Microsoft **SQL Server 2005** which is one of the top 3 most used in the IT business. [7]

The persistence layer was handled with **NHibernate**, an ORM tool that supports the DDD (Domain-Driven Design) approach used for modeling the domain.

MSMQ (Microsoft Message Queuing) was chosen as the MOM implementation for its availability in every Windows Operating System since version 95.

WCF was used for the Web Service implementation as it can be easily deployed dynamically, inside a service container.

For managing HL7 messages, the only open-source library found was **NHAPI**, even though it has several limitations (doesn't include the communication protocol and hasn't been updated for a long time).

DICOM messages and objects are managed with the recent (but good) **mDcm** library that includes a code base good enough to implement any DICOM service on top of it.

Experimental results

In order to evaluate the qualities of the HIE, some experimental tests were conducted.

The **reliability** of the system's components was measured using the MTBF (Mean Time Between Failures) formula:

$$MTBF = \frac{total\ uptime\ (h)}{number\ of\ failures}$$

The values of the total uptime were determined from the service and service container event logs in a testing environment where stable prototypes are usually deployed and tested with real-world healthcare-related applications. Only the failures that actually caused service or service container downtime were considered.

The results of these tests were the following:

Tested component	**# of instances**	***Total uptime (h)***	**# of failures**	**MTBF (h)**
Service Container	2	7536	2	3768
Transformation	8	6826	0	n/a
CBR	1	57	0	n/a
HL7 Receiver	1	1248	1	1248
HL7 Transmitter	2	3955	0	n/a
DICOM Worklist SCP	1	3479	3	1160
DICOM MPPS SCP	1	1545	1	1545
DICOM Q/R SCU	1	599	11	54
Database	3	3194	0	n/a
Web Service	7	5833	2	2917
Web Service Caller	2	936	0	n/a

Table 1 - MTBF Results

Most of the failures than can be observed in this table were related to bad exception handling in child threads of the services and all these problems have already been solved.

On the other hand, the case of the DICOM Q/R SCU can be justified by the fact that the testing environment is the only one where this service was effectively tested because of the lack of testing tools. Since all the initial testing of this service was done here, no wonder the number of failures is so high.

The **efficiency** of the system's components was measured by the number of messages each service can process per second with approximate stress testing approaches.

The tests were conducted in the development environment using two PCs:

1. PC with an Intel Core 2 Duo processor @ 1.86 GHz and 4 Gb of RAM memory, using the OS Microsoft Windows Vista Business SP2 (64-bit) and the .NET framework 3.5 SP1;
2. PC with an Intel Pentium D processor @ 3.0 GHz and 2 Gb of RAM memory, using the OS Microsoft Windows XP Professional SP3 (32-bit) and the .NET framework 3.5 SP1;

The first machine was always the host of the services being tested. When external testing tools were needed, like Interfaceware Chameleon HL7 Simulator/Listener or OFFIS DCMTK Query (C-Find) SCU, the second machine was always the one used for hosting these tools.

The service DICOM MPPS SCP wasn't tested owing to the lack of testing tools, while the services DICOM Q/R SCU and Web Service Caller were also not tested because they were fully developed by the time the tests were conducted.

The external testing tools were used to test and measure the performance of both the HL7 services and the DICOM Worklist SCP service with 500 hundred messages, splitting the load in threads when possible (concurrency testing).

The Web service was tested with a test client invoking it in 500 background and concurrent threads.

The other services were tested by injecting 500 messages directly into their message queues.

All the tests were conducted at least three times and the mean value of the results is presented in the following table:

Service	**# of messages**	**Processing time (sec.)**	**Messages/ sec.**
HL7 Transmitter	500	16,858	29,659
HL7 Receiver	500	120,624	4,145
DICOM Worklist SCP	500	185	2,703
Web Service	500	7,904	63,259
Database	500	3,159	158,278
DICOM MPPS SCP	-	-	-
DICOM Q/R SCU	-	-	-
Web Service Caller	-	-	-
Transformation	500	208,24	2,401
CBR	500	1,868	267,666

Table 2 - Stress test results

Most of the results are either within or above the expected ranges.

The DICOM Worklist SCP is one of the slowest, but this isn't a problem because it is a type of service that will typically receive no more than 100 requests per day.

The results from the HL7 Receiver were disappointing. This service is using a custom parser to translate HL7 messages to XML which hasn't been optimized yet, so more work is definitely needed.

The results from the Transformation service were expected since XSLT transformations are computationally heavy. The recommendation in this case is to distribute transformations along the available Service Containers to maximize the hardware resources.

Conclusions

The main goals of this work were achieved with the development of the HIE. The architecture chosen for the system has been a success, allowing the dynamic deployment of integration services in the service containers and letting the user distribute the services according to the available hardware resources.

The system was developed in a modular fashion, making it easy to change a part of the system without compromising the rest. There are also several extensibility points that can be used to improve the system's features.

One of the defining points of the system is the Base Service that provides an easy integration with the rest of the system, allowing a developer to concentrate only on the new features that add business value to the system.

Although the HEI is meant for healthcare environments, the core of the architecture is generic enough to be used as a part of any integration engine.

Comparing the features of the HIE and some of the existing solutions in the market, we can conclude that this system can already compete with several others in many of the features, although the support for some commonly supported standards is still missing.

The support for other standards is probably the most relevant future work that can be done to improve the HIE.

Bibliography

1. KHOUMBATI, K.; THEMISTOCLEOUS, M.; IRANI, Z. Evaluating the Adoption of Enterprise Application Integration in Health-Care Organizations. **Journal of Management Information Systems**, Volume 22, Issue 4, 2006. 69-108.

2. GEE, T. Current State of Medical Device Connectivity. **Medical Connectivity**, 13 Junho 2005. Disponivel em: <http://www.medicalconnectivity.com>.

3. MOTOROLA. **Enterprise Integration: A Key To Successful Public Sector Customer Service**. [S.l.], p. 3. 2003.

4. CHAPPELL, D. **Enterprise Service Bus**. Sebastopol, EUA: O'Reilly, 2004.

5. BUSCHMANN, F. et al. **Pattern-Oriented Software Architecture**. [S.l.]: Wiley and Sons, v. 1, 1996.

6. HERNAN, S. et al. Uncover Security Design Flaws Using The STRIDE Approach. **MSDN Magazine**, Novembro 2006.

7. MARKET MAGIC RESEARCH. **2005 RDBMS Software Cost of Ownership Study**. [S.l.], p. 15. 2005.