“Event-Driven” Health Care Information System

A.F. Dragoni[1], D. Faina, G. Giampieri[2]

[1]DIIGA, Università Politecnica delle Marche, Ancona, Italy
[2]ASUR Zona Territoriale 7, Ancona Italy,

Abstract

This paper introduces an event-driven architecture for health care information systems. It can solve the problem of interoperability among the different software in a complex health structure.

Introduction

The heart of the clinical process is the medical information and it is just for its importance that it should "survive" outside the system who created it and it is essential that it is made available to whom need it for a treatment.
Throughout the ages we attended the introduction of many health information systems: it has been a tumultuous, sometimes wild growth process.
These systems had the purpose to satisfy the specific operative and managerial needs of important realities in the health environment. These realities, however, were limited, so their implementation had been done without any intent to integrate them with the systems already existing in other services. For this reason the digital information produced in laboratories, in diagnostic services, in departments or ambulatories are not shared or distributed, they do not become the common heritage of the whole health/sanitary organization, in order to be accessible wherever and whenever necessary to the various operators, but they remain confined in this computer systems which had produced them.
We have also to say that not all the players of the health service use the magnetic support for the informations recording, particularly in the departments. This is due to the lack of cheap instruments that allow you to gather data directly from the patient’s bed without using paper and to the difficulty of standardizing the vocabulary to describe the informations gathered. In this way, the informations could be placed and found without any problems.
To solve this situation, the key word is: interoperability, which includes:

1. dissemination of ICT standards in the health system;
2. dissemination of integrable and not integrated softwares.

Standards in healthcare

The most important standards in the health sector are certainly the DICOM and HL7.
The DICOM standard specifies how data relating to diagnostic tests such as CAT, NMR, ECG, etc. .. should be structured.
Till now this standard has been used only for the management of radiological images and in this ambit we witnessed an incredible improvement of interoperability between PACS and RIS systems, intra-extra structure.
The HL7 is a standard, it is at level 7 of the ISO-OSI hierarchy and it specifies the semantic structure of the messages that must be exchanged in the health ambit.
The final version, the 3rd, aims to introduce the following changes as to the previous version:

- To adopt an Object Oriented approach for the development of the standard, which is the same used for the development of DICOM, based on the standard Unified Modeling Language (UML). This change increases the detail, the clearness and the accuracy of the specifics, favoring both the actors that control the development of the standard and the developers who should implement the same standard.

- To reduce optionalities. The optionality in version 2.x is a stimulating strong point for the diffusion of the standard, but it is also a weak point at the same time. For this reason, the version 3 has as object the reduction of the optionalities number, with the consequence of an increase in the number of messages.
- To make verifiable the compliance to the standard. It's hard to measure the conformity of the interfaces HL7 version 2.x. The retailers state the compatibility without having any formal documentation which demonstrates how and with what are they compatible, because the standard itself does not provide any support.
- To release the specifics of technology defining in an abstract way the messages structure.
- To provide a framework (explicit model + architecture) to match events, data and messages.
- To improve the clearness and the accuracy of the specifics.
- To improve the adaptability of the standard to changes.
- To exploit emerging technologies such as XML.
- To move towards "plug and play"direction.
- To single out models as a collection of subject areas, scenarios, classes, attributes, use cases, actors, trigger events, interactions, which determine the necessary informations to specify the messages of HL7 version 3 [1]

The IHE [2] authorized by the organization that manages the HL7, attended to define what should be the HL7 v2 messages, what should be their structure and when they should be sent among the different systems managing the health services. This IHE works can be adapted to version 3 of the HL7 standard.
The reduced use of this standard, however, is connected with the fact that the softwares for health care continue to be designed to work independently.

The event driven structure in health care

A good software for health care should adapt itself as much as possible to this environment and it should not happen the contrary.
The health care sector is very dynamic and this is due to the fact that often the cases are different and for this reason the procedure followed is not always the same, we can say that in this sector the evolution of situations is characterized by the flow of events which are completely unknown a priori.
The aim of this paper is to propose an event driven information health care system that solves the problems mentioned above.
An event driven structure is typically formed by events notifiers, a dispatcher and notifications receivers. The receivers have a precise duty: to react as soon as possible to the arrival of a notification.
The dispatcher is responsible for dispatching the events of the notifiers to the appropriate receivers. The reaction may be provided entirely by a single receiver but the receiver itself may change the notification format and forward it to another receiver delegating him the responsibility of the execution. The basic concept of this structure is that once the producer send the notification, he ignores the subsequent processing.
The event driven architecture can be seen both as a programming paradigm and a method for managing a distributed computer system.

Event driven programming paradigm

In a programming it is often necessary that a particular sequence of actions is repeated with different values of the data on which it operates.

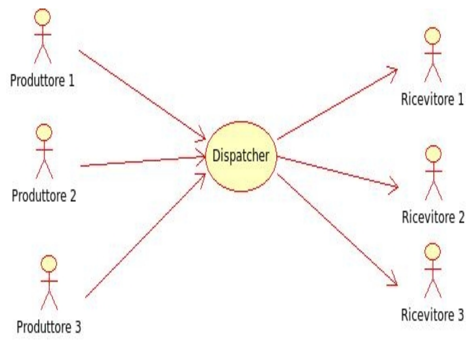
Now we consider for example, the trigonometric operation of the sine. Not all architectures have instructions that calculate the sine. In this case the programmer builds a “subprogramme” that will be named for example sin and it will be called whenever there wiil be necessary to calculate the sine. In the “subprogramme” will be passed as parameter, the value of which you want to calculate the sine. In C you would write:

```
y=sin(x)
```

where x is the variable of which you want to calculate the sine and y is the variable to which you assign the calculated value.
In terms of machine language operations, the “subprogramme” represents a block of instructions and the call is a transfer of control to the subprogramme. Since the subprogramme must be called by any part of the programme, it is necessary that when it ends the control, it returns to the point where the call was. For this reason it is necessary to establish the mechanisms and the conventions with regard to the way to:

- to make the connection among programme and “subprogramme”;
- to keep mark of the return address;
- to execute the passage of parameters.

The control transition from the calling programme to the subprogramme requires a specific jump instruction. This instruction has to modify the contents of the PC register (programme counter) with the address of the first subprogramme instruction and it should also save the address of the instruction which follows the one that made the call in the calling programme (context switch).
However there is also another way through which it is possible to change the flow of a programme: the interruptions. An interruption obliges the CPU to temporarily stop the execution of the current programme to pass to another programme which is dependent on the nature of the interruption.
To understand the nature of the event driven paradigm we may say that if we want to implement it with an embedded system we would act on the interruptions, in other words, any kind of event would break off the normal flow of a running programme to make room for a new specific elaboration.
In the case we are in another context, different from that of an embedded system, we must use the instruments that the various programming languages make available to realize this architecture.
We can say that in the event-driven programming paradigm, unlike call-stack paradigms (described above), there isn't the concept of a method that invokes another method and consequently do not exist the concepts of coordination, continuation and contextualization. The coordination consists of the synchronous execution of actions, in other words: the calling method waits for results from the method called before keeping on with the execution. The continuation ensures that when the method called reprunes the results the operations will start again from where they stopped with the call. Contextualization means that local variables are saved as part of the implementation context and once the method invocation will be finished the whole context will be restored.
The interaction among the components in the event driven paradigm is limited to the fact that a component sends an event and one or more components receive it.
Now we are going to see the use case diagram which describes this paradigm:



In the picture above we can see three components: the events generators, the events users and the dispatcher. The duty of the dispatcher is to take each event that arrives, to analyze it to determine of what kind it is and then to send these events to receivers that will be able to work out events of that kind. The dispatcher must be based on a loop that allows it to continually repeat the operations of reception and distribution of events but only if their type is one of those the dispatcher is able to manage otherwise the events will be refused.
The pseudo-code of a dispatcher is the following:

```
do forever:     # the event loop
    get an event from the input stream
    if    event type == EndOfEventStream :
          quit # break out of event loop
    if    event type == ... :
          call the appropriate skin,
          passing it event information as an argument
    elif event type == ... :
          call the appropriate skin,
          passing it event information as an argument
    else:    # handle an unrecognized type of event
          ignore the event, or raise an exception
```

If the pseudo-code above should be implemented in C each call would be done with pointers to functions or callback function used asynchronously, therefore first they will be registered and then called. In some cases, the dispatcher and the receivers are not able to manage events so quickly as they arrive. In this case the events at their arrival are placed in a queue. The dispatcher therefore will draw the events from this queue and then it will deliver them to the receiver with the appropriate speed.
To sum up, we may say that the event driven programming paradigm is particularly advantageous compared to the call-stack when the following conditions does not occur:

- something happens in a precise moment;
- we know what happens and in what order it happens;
- we know anyone who can provide to solve a specific task. [3]

The event driven architecture for managing distributed systems

Now we will see how the event-driven architecture may be useful help to promote interoperability among informative systems in an healthcare structure.
The events in this context, will be represented by everything that happens to a patient: hospitalization, discharge, performance of a CAT, of a magnetic resonance, of a laboratory analysis ....
The producers of events notifications are consequently the management software of individual departments, the RIS, the PACS and the other laboratory systems. The users of notifications coincide with producers adding the general practitioner’s software, the administrative system, the Registry office and the CUP (the department software receives the events from the RIS, the PACS receives the events from the RIS, the RIS receives events from the PACS, the general practitioner’s software receives the events of hospitalization and discharge from the department software, the management system receives the events about the execution of an examination from RIS, the department software receives the events generated from the registry office, the RIS receives the events generated by the CUP ...).
It would be unthinkable to handle this system with blocking mechanisms because if, for example, the department system to which the RIS must notify an event is disconnected, the radiology system would be blocked until the function is performed.
All this could be avoided by using an events notification system that functions as the dispatcher of the event-driven programming paradigm, so when it receives notifications from producers, it

analyzes them and it distributes them asynchronously to all the users interested in that type of event. In this way the producer resumes its activities, once generated the event and sent the notification to the dispatcher. If the event notifier is constructed with a non-blocking structure it can be overcome the problems of connection with the dispatcher (eg managing the sending of notifications with a callback function which is executed asynchronously, if the dispatcher is on line the function indicated by the pointer to send the event, is immediately carried out, otherwise the parameters to be passed will be recorded in a queue that will be managed by an external component that consists of a loop. The loop will check if the dispatcher is connected and in that case it will send the messages to the dispatcher [4].
To ensure that messages are actually delivered to various parties interested we need to use the "acknowledgment" message that will be sent by the dispatcher to the generator of the message and by the receiver to the dispatcher. Always asynchronously and non-blocking.

Implementation

In a event driven structure it is essential to determine how to establish communication.
For the "transport system" there are several possibilities: HTTP, FTP, HTTPS.
For the formatting of the messages body the obvious solution seems to be XML and it is what we stand for. With regard to the semantic level we refer to the specific HL7 and DICOM.
The dispatcher is a crucial crossroads, because it allows us to handle the access to informations, which are essential in the healthcare sector.
A good dispatcher should implement the point to point communication domain and the publisher / subscriber domain. In the first case we have a channel between the producer and the receiver of the message, in the second case, we have a topic in which many users will enrol, some as generators of events and some others as events underwriters, in this domain an event is forwarded to all the registered receivers.
It is important that the dispatcher in both domains involves the use of queues to asynchronize the notification process.
The dispatcher prototype conceived in this study has been implemented through a series of Linux shell scripts that are able to receive messages, analyze them, and according to their type send them to different queues from which they will be asynchronously sent to the receivers with different communication protocols.

Conclusions

An event driven system seems to be a good solution for interoperability, the development of HL7 and the construction of terms codification used to describe the medical informations.
Additionally, this system seems to be optimal for a over departmental health dossier. In fact with a simple change of the dispatcher, certain events are recorded in a database that will be accessible only to authorized persons. To facilitate the selection of events that will be inserted in this dossier, and also their discovering, it could be structured as a history case oriented to problems, in which each patient is characterized by problems (active or not) and to each problem (identified by the ICD code of the main patient’s disease) are associated all the events organized in 3 subclasses: results of diagnostic tests, diagnosis of secondary disease and medical treatment.

References

[1] "InterSem/HL7: progettazione ed implementazione di un sistema di interoperabilità semantica per lo standard HL7 v.3" - Politecnico di Milano - A. Carenini, Francesco Pinciroli, Dario Cerizza
[2] http://www.ihe.net/
[3] "Event-Driven Programming: Introduction, Tutorial, History" - eventdrivenpgm.sourceforge.net
[4] “Programming Without a Call Stack – Event-driven Architectures” - eaipatterns.com/docs/EDA.pdf
[5] "Cartella clinica orientata per problemi: strumento per superare possibili conflitti tra diritto alla salute e diritto alla riservatezza?" Cristiano Cortucci, Marco Valsecchi - www.medleg.it/CORTUCCI-CartellaClinica.pdf