Revista Antioqueña de las
Ciencias Computacionales y la Ingeniería de Software

ISSN: 2248-7441

www.fundacioniai.org/raccis
raccis@fundacioniai.org

RACCIS

Refactoring-Based Design of Reference Architecture

Diseño Basado en Refactorización de la Arquitectura de Referencia

Francisca Losavio[1], Oscar Ordaz[2], Victor Esteller[3]
[1,2] Universidad Central de Venezuela. *francislosavio(AT)gmail.com, oscarordaz55(AT)gmail.com*
[3] Universidad de Carabobo. *vesteller@gmail.com*

INFORMACIÓN DEL ARTÍCULO

Tipo
Investigación

Historia
Recibido: 21-01-2015
Correcciones: 18-05-2015
Aceptado: 05-06-2015

Keywords
Software Product Line; Reference Architecture Design; Reference Architecture; Refactoring Process; Quality Model; ISO/IEC 25010; Softgoals Interdependency Graph; Integrated Healthcare Information System

Palabras clave
Línea de Productos de Software; Diseño de Arquitectura de Referencia; Arquitectura de Referencia; Proceso de Refactorización; Modelo de Calidad; ISO/IEC 25010; Grafo de Interdependencia de Softgoals; Sistema de Información de Salud Integrado

ABSTRACT
A Software Product Line (SPL) is a set of software-intensive systems, sharing a common, managed set of features that satisfy the specific needs of a particular market segment or domain. These features are developed from a common set of core assets, which are reused in different products or software systems that form a family. Reference Architecture (RA) is the main asset shared by all products of SPL; it covers commonality and variability of the SPL family of products and it is used as a template to produce new products in an industrial production context. Responding to industrial practice, in previous works we have proposed a semi-automatic bottom-up refactoring process to build RA considering the architectures of existing products, represented by a connected graph or valid architectural configuration; in this paper a new refactoring process is proposed, considering variability modelling of components and connectors, which had not been deeply treated in our previous works. On the basis of the domain products' semantics similarity of components, a candidate architecture (CA) is obtained automatically and it is manually completed into the final RA using the ISO/IEC 25010 standard quality model combined with goal-oriented techniques to consider also non-functional variability, which is still an open research issue in SPL. Finally, RA is manually constructed from the completed CA by grouping the variants obtained performing similar tasks into variation points. The RA design refactoring process is applied to the domain of Integrated Healthcare Information Systems. A prototype tool supporting the proposed process is under design.

RESUMEN
Una Línea de Productos de Software (LPS) es un conjunto de sistemas intensivos de software que comparten un conjunto organizado de características que satisfacen las necesidades de un segmento particular del mercado. Estas características se desarrollan a partir de un núcleo común de activos o "assets", que son reutilizados en diferentes productos o sistemas de software que conforman una familia. La Arquitectura de Referencia (AR) es el principal activo compartido por todos los productos de la LPS; cubre las partes comunes y variables de la familia de productos y es utilizado como una plantilla para producir nuevos productos en un contexto de producción industrial. Respondiendo a la práctica industrial, en trabajos previos hemos propuesto un proceso semiautomático descendente de refactorización para construir la AR considerando las arquitecturas de productos existentes, representadas por un grafo conexo o configuración arquitectónica valida. En este artículo se plantea un nuevo proceso que considera en mayor profundidad el problema de la variabilidad de componentes y conectores, el cual no había sido abordado en los trabajos anteriores: sobre las bases de la similaridad semántica entre components de productos en un dominio dado, una Arquitectura Candidata (AC) es obtenida automáticamente y es completada manualmente para obtener una AR final, utilizando el modelo de calidad estándar ISO/IEC 25010, junto con técnicas de orientación a metas para tratar también la variabilidad no funcional, la cual es aún un tema de investigación abierto en LPS. Finalmente la AR se construye manualmente a partir de la AC completada, agrupando las variantes obtenidas que realizan tareas similares en puntos de variación. El proceso completo de diseño arquitectónico por refactorización es aplicado para obtener una AR en el dominio de los Sistemas de Información Integrados de Salud. Una herramienta prototipo de apoyo al proceso propuesto está siendo diseñada.

1. Introduction

A *Software Product Line (SPL)* is a set of software-intensive systems - or complex evolving systems - sharing a common, managed set of features that satisfy the specific needs of a particular market segment or domain. These features are developed from a common set of core assets, which are reused in different software products that form a family [1].

The *SPL* approach favours reusability and claims to decrease costs and time-to-market. The key issue in *SPL* development is the construction of a common architecture from which new products or software systems can be derived. Our design process is framed in the context of *SPL Domain Engineering* [2]. A *Reference Architecture (RA),* often found in the literature as Product Line Architecture, is a generic architecture for high-level design of *SPL* products; it considers a *core* or set of components common to all products of the *SPL* family and a *variability model* to indicate and document how a component can be customized or instanciated to derive new products [2, 3]. *Software Architecture* is defined in [4] as "a collection of computational components - or simple *components* - together with a description of the interactions between these components, the *connectors*".

In this work, the refactoring (also called reactive/extractive) *bottom-up* approach [5, 6, 8] is followed to construct *RA* because in practice, many industrial organizations do not dispose of such a framework or generic architecture. In the bottom-up refactoring context, existing similar products (having similar architectural style and/or functionality) must be examined, using reverse engineering techniques, to identify commonalty and variation points. The *RA* design is a complex process that is in general poorly described in the literature and left to incomplete case studies; details of methods and approaches are difficult to follow because there are no design standards [6, 9]. Moreover, existing traditional architectural methods and evaluation techniques for single-systems are reengineered and not specifically designed for *SPL* [7]. In the reactive approach, domain and architectural knowledge is recovered from existing products [5, 6].

An important question that commonly appears in the literature is what needs to be done to ensure a suitable choice of architecture for the *SPL* family of products. To answer this question, this work provides a bottom-up refactoring process, founded on a graph structure to model the product's architecture, to assure connectivity of an architectural configuration; our process produces automatically an initial architecture or Candidate Architecture (CA), considering semantically similar components; functional and non-functional variability can eventually appear in this initial version; *CA* is then completed with new components introduced to satisfy non-functional requirements, using goal-oriented techniques [16] combined with a standard hierarchical specification of non-functional requirements, expressed as quality requirements using a quality model [11]. The product architecture is represented by a *connected graph* or *valid architectural configuration (P,R)*, where *P* represents *components* and *R* is a symmetric relation between each pair of components representing *connectors*.

A component provides/requires interface to/from another component; in this work other relation types among components and ports will not be considered, and only the logic view [12] of the architecture expressing static aspects, specified using UML 2.0 [13], will be used. The given products' commonality kernel or *Common Core (CC)* is defined as the set of common components present in each given product architecture. The initial *CA* is automatically produced, performing the union of the components and connectors of the architectural configurations of each given product, after a similarity analysis and component names' unification, and preserving the connections.

The goal of this work is then to propose a refactoring process to design the *RA* for the *SPL* from existing products; this process is inspired in [10, 48] and improves the *RA* design considering a documented and justified variability modelling of functional and non-functional requirements, including connectors, by combining the goal-oriented approach using the *Softgoal Interdependency Graph (SIG)* [14, 15] and the *ISO/IEC 25010* standard for software product quality modelling [11]; notice that in the literature the terms non-functional requirements, quality requirements, quality attributes, quality properties and quality characteristics are often confused; in our case, according to [11], a non-functional requirement is the textual expression of a requirement and the quality requirement is a property or characteristic that can be measured on a quality attribute to express the requirement's quality goal.

One of the problems with the *SIG* is the lack of standard notations and the handling of complexity of the graph configuration; however, even if automatic tools are available to support the SIG construction [16], few guidelines are provided on how to make a "good" decomposition or refinement of quality requirements [48]. In our case, *SIG* will help to trace the quality requirement and retrieve the responsible functional component, improving requirements traceability, and to model variability introducing new components to solve quality requirements, and also to detect inconsistencies in configurations, such as the presence of conflicting components. The basic terminology of Pohl et al. [2] is followed for the variability modelling, where *variant* or *variability object* is the "instance" of the *variation point* or *variability subject*; a *context* is associated to the variation point for documentation purposes. For example, "internet" or "satellite" are objects of the "network" variation point and its context could be the "information transmitted". However, the variability of connectors between variations points is nor treated; in this work, this issue is considered as part of the variability model.

Since our process is bottom-up, the study of existing products in the domain should be sufficient to derive the *RA* using reingeneering techniques; however, we consider that a quick study of the domain, or Domain Analysis in the literature, considered the first step of Domain Engineering, will facilitate the capture of global domain knowledge, such as architectural style(s) used, the domain main functional requirements, and non-functional requirements with their quality requirements, that in our case will be specified by the standard quality model. In this sense, our work stands as a middle-ground between the SPL classic top-down approaches that capture general domain knowledge to construct a feature model [19] to derive the RA [6], and the bottom-up approach, that builds the RA based on the knowledge obtained from the refactoring of existing products' architecures [10, 37]. This work proposes to apply the *RA* design refactoring process to a case study in the *Integrated Healthcare Information Systems (HIS)* domain, focusing on free-use software of the Open Source Foundation (OSI).

This paper is structured as follows, besides this introduction: the second section presents a brief Domain Analysis, which in our case concerns the *HIS* domain, including a discussion on the standards involved in our approach and the similarty analysis of products' components. The third section is dedicated to the *RA* refactoring design process, which is applied to the *HIS* domain in the *SPL* context, taking the *HIS* domain as a case study. The fourth section presents an example of instantiation of *RA* to derive new products' architecture. The fifith section discusses related works. Finally, the conclusion and perspectives are presented. An Appendix is provided, containing definitions and general metrics of the quality characteristics of the standard quality model [11], customized for the *HIS* domain.

2. Domain analysis: Healthcare integrated information systems

A clinical record or *Health Record (HR)* is the set of documents containing data, values and information on the situation and clinical evolution of a patient over the whole assistance process. Privacy, availability and persistency are priority quality requirements associated to systems supporting the management of these kinds of documents. *HR* evolves into the *Electronic Health Record (EHR)* with the introduction of information and communication technology. *EHR* management is now one of the major functionality of an *Integrated Healthcare Information System (HIS)*.

The term "integrated" means that telemedicine (online healthcare services) is generally supported with the possibility of accessing healthcare data from different electronic patient record systems and other systems. A modern *HIS* makes sense only if interoperability of *EHR* among regional, national and international healthcare institutions can be guaranteed. The *RA* for *HIS* has to meet main quality requirements, centered on *EHR* management issues, specified as a *HIS Quality Model (HIS-QM)*: *security* for *EHR* (*authenticity* for authorized access control, *confidentiality* or *"privacy"* for the user's rights policy, and data *integrity*), *reliability* (*availability-persistency*, since *EHR* must be "always" available), *portability* (*adaptability-scalability)* to different platforms and data volume, and *maintainability* (*modularity, modifiability)* for system evolution including management of medical standards; we will include also *efficiency (time behaviour)* since time is crucial for quick medical assistance and information retrieval; *functional suitability* (*correctness* or *"precision"*) is required for certain computations in reports and medical orders.

The architecture is in general responsible for these non-functional requirements. However, the *interoperability* of *EHR* depends on the standards used for the internal structure of the document and on the way it is transmitted; on the other hand, since *EHR* cannot be destroyed, their *persistency* depends on how this feature is implemented by the repository or database used; *availability,* instead, is appreciated in data transmission. Figure 2 shows the standard terminology used in ISO.IEC 25010 [11]. The main *HIS Functionalities* found are: *management of EHR, queries of medical information for correct diagnosis, edition of orders for medical treatment, access and consult of patient demographic data, and patient attention services*. Other functionalities are also found in some *HIS*, such as management of medical imaging, hospitalization and surgery rooms facilities; we are considering here only the most common functionalities.

2.1 Standards used

Standards are documents established by consensus and approved by a recognized organism; they provide rules, guidelines or characteristics applicable for a repeated use of its activities or results. In the *HIS* case, the use of standards is mandatory to guarantee interoperability of medical data. Work has been going on from about fifteen years on standards for the healthcare domain, nevertheless the community has not yet reached an agreement [22, 23]. In what follows, we will be limited to mention those standards directly involved in the *HIS* products considered, and the standard used here to specify the software product quality.

HL7 (Health Level 7) version 3 (2003) messaging standard to support communication between the healthcare institution and medical records (Health Care Records (HCR) or Electronic Medical Records (EMR)) produced by different *HIS* [24]. *HL7 CDA (Clinical Document Architecture)*, is proposed as an ANSI standard in 2005; offers 4-layers: GUI, service, logic, data (persistency). It contains specifications of message formats, electronic document structure and vocabulary for the healthcare domain. It has been approved as an ISO standard [24, 25].

HXP (Healthcare eXchange Protocol) is a standard data exchange protocol used in the healthcare domain for transparent communication among institutions, independently of the platform. It consists of an XML message format and a Procedure Call Dictionary (PCD) [35]. *OpenEHR:* Open standard to create normalized *EHR*, knowledge management oriented. It offers two separated models (dual models) favouring modularity, low coupling and reuse; they allow semantic interoperability among different healthcare actors: - information model or generic reference model for the software component, and -

knowledge model (archetypes and patterns) separated from software, now part of the ISO 13606 standard; the specification is written in UML [26, 27].

Software quality models have been widely used in software engineering practices to measure the quality of a software product [11, 47]. One of the best kmown and used is the *ISO/IEC 25010 quality model* [11], see Figure 1, which provides a hierarchical model describing a set of eight high-level quality characteristics that are refined in multiple levels to define the product quality, until the measurable elements, called attributes, are reached. It offers two quality models: 1) product quality (internal/external), perceived by stakeholders during the product development process, and 2) in use quality (perceived on the running system by the end-user). A product can also be a software intermediate artefact produced during the development process, such as the use-case model or the architectural configuration. In this work we will use this standard to specify the product quality model.

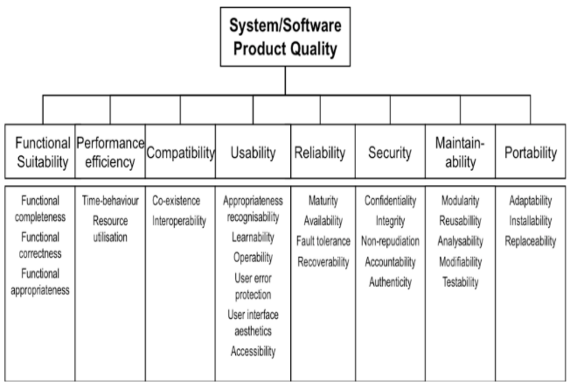


Figure 1. ISO/IEC 25010 Product Quality Model [11]

2.2 Description of the *HIS* architecture.

Many commercial and open source *HIS* are commonly available for main informatics platforms, however the majority are not compatible or do not handle *EHR*. Moreover, their complete adoption by healthcare staff finds cultural, social and organizational obstacles [28]. According to the study in [29], the open source systems *OpenEMR* [26] and *PatientOS* [31] present a 90% and 92% usage respectively, according to 2010 data.

OpenEMR is an Electronical Medical Record (EMR) management Web-based system, offering usual *HIS* serveces for geographically distributed clients and different platforms, assuring portability and interoperability through Internet, supporting also mobile devices networks. *PatientOS* is based on the OpenEHR standard, following a client/server classic distributed model of 3-layerd information systems; it holds an MVC-based Graphical User Interface (GUI) to ensure maintainability and it retrieves remote services from a Web Server through the process layer. *GNU Health* (2008) [50], based on the three tiers Tryton 3.2 architecture for an *ERP (Enterprise Resource Planning)* system, has arised as a prized *HIS* and has been recently adopted in several countries as the official e-health system; GNU Health supports HL7 version 4, called FIHR (Fast Healthcare Interoperability Resources) and combines *EHR* with special provisions for small hospitals and clinics; its architecture is similar to PatientOS, supporting an Model, View, Controller MVC-based GUI on the client, for modifiability and scalability; on the server it has a security component to handle confidentiality and authenticity, not limited to SSL (Secure Socket Layer).

PatientOS and GNU Health follow a client/server model and the Process Layer can be integrated to the Web to retrieve Web services by a Web Server like for example Apache, communicating through a gateway; however even following a *Service-Oriented Architecture (SOA)* [34], they are not web applications, since their GUI has two separate components, classic multi-windows display on the client in the Presentation Layer and logic (GUI server-side) running on the server in the Process Layer; GUI communicates with the Precess Layer via tcp/ip by RPC (Remote Procedure Calls). GNU-Health is similar to PatientOS. Another open source system used in recent concrete national projects was *Care2x* [28, 32, 33], very similar to OpenEMR. The architecture of these three systems was studied on the basis of the general available documentation found on-line.

Figure 2 shows a general hybrid architecture SOA+Layers used in the *HIS* domain; notice that the UI component in the Presentation Layer is present in PatientOS (b5. GUI Server-side) and absent in OpenEMR and Care2X; the LAMP platform. Linux OS, Apache, MySQL, PHP is followed by both OpenEMR and Care2X, to integrate the classic 3-tiers (presentation, process and data) architectural style; the communication/transmission tier crosscuts all other tiers, and it is supported by the Apache Web Server and an additional integration tier is added to achieve *EHR* interoperability with the HL7 standard; SSL is the Security component (HTTPS).

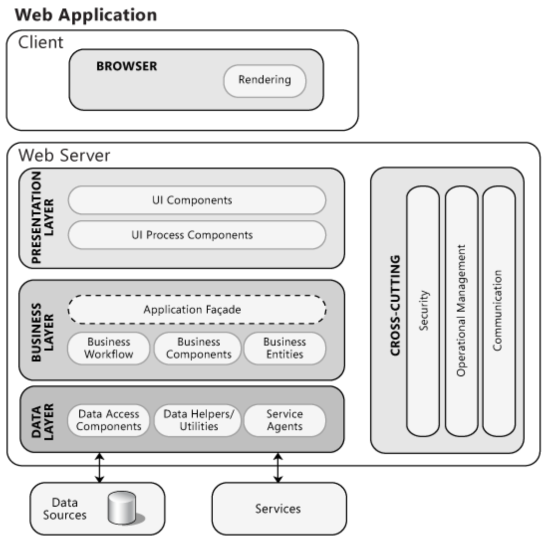


Figure 2. General hybrid architecture client-server/SOA +Layers, used also in the *HIS* domain (image from Wikipedia)

In conclusion, we observed two main categories of *HIS* systems, according to the user interface (UI) configuration: Web-pages-based systems retrieving services through a Browser in the Presentation Layer, under *SOA* (like OpenEMR and Care2x) and classic GUI-based systems, retrieving services at Process Layer level, connecting to a Web Server through a gateway (like PatientOS and GNU Health); all the systems studied follow a distributed client/server/SOA model combined with the classic layered style of information systems: *Presentation Layer (a), Process or Business Logic Layer (b) and Data Layer (c),* in addition to the *Communication/transmission Layer (d)* including all the information transport on the networl *(d1, d2)* (see Table 1 unifying the component names of the products); this layer ensures portability (adaptability-scalability) and maintainability (modularity, modifiability) with respect to Web services, as well as the distribution to geographically distant locations by Internet, and locally within a healthcare institution by Intranet; Security (authenticity, confidentiality or "privacy" and integrity) of *EHR* document transmission are also assured by the *d. Transmission Layer*.

Other differences are on how to solve interoperability (components *b4, c2,* and *c3*) and adaptability with respect to different databases, not shown in general in the products graphical description of the architecture; separated GUI (*a4-b5*) components in PatientOS and GNU Health ensure more system modularity and modifiability, not limited to Web services maintainability, offering greater availability because it does not depend on the availability of Web Servers. The dynamic Web-pages UI (*a1*) common to OpenEMR and Care2x, is of low cost and fast development, offering however less availability, for the above discussion. Data persistency and integrity are assured by the database (*c1*); some of the systems assure also portability to different databases: Care2X supports MySQL and PostgreSQL (API ODBC for portability), PatientOS supports MySQL and Oracle (API JDBC for Java portability), and OpenEMR supports only MySQL. In this study, the architectures of OpenEMR, PatientOS and Care2X were considered; GNU Health, very similar to PatientOS, has been left out to ease the presentation; Figure 3 shows the architectures of the three products expressed in UML 2.0.

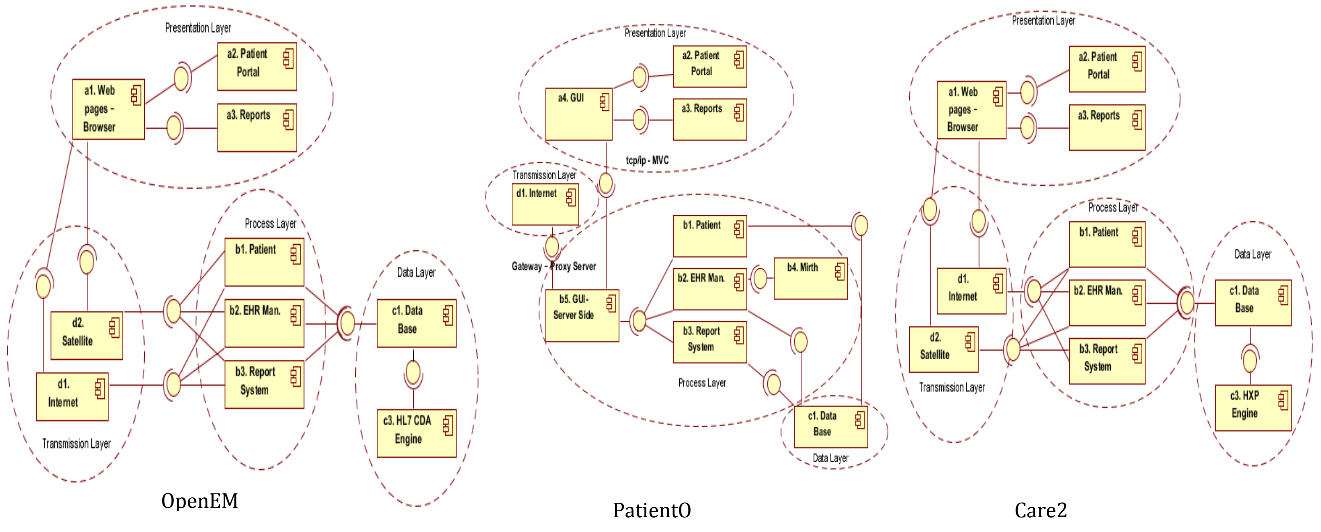


Figure 3. Three *HIS* architectures: OpenEMR, PatientOS, Care2X

2.3 Analysis of similarity

Semantic similarity of components has to be studied in each product to determine the functionality common core. Table 1 presents the architectures' components after an unification process of the components' names, respecting the semantics of the functionalities as far as possible, on the bases of the available documentation; these components are the input to the bottom-up refactoring process of the *RA* design that will be presented in the next section. Connectors between two components *x, y* are denoted by *xRy*, *R* a symmetric relation. In Table 1, the "-" symbol indicates the absence of a specific connector, since one of its component is absent or the absence of a component. The "X" symbol indicates that components are present, but their connector is absent in a product. Common and variant components and connectors are also shown.

The main *HIS* functionalities, identified in the three systems for the purposes of this work, were: - *basic services for patient attention including scheduling, demographic data, etc., - EHR management (b1, b2), accessed through a "Patient Portal" (a2),* and - *"Reports" (a3)* allowing to access medical and administrative facilities *(b3).* Other functionalities have been identified, such as help to medical diagnosis, imaging, hospitalization and operating room facilities, but we have limited this presentation to the most representative ones to illustrate our approach.

3. Refactoring process for RA design

The proposed *RA* design refactoring process will be applied to the *HIS* domain and considers the following basic steps.

3.1 RA Design Refactoring Process

Basic input from Domain Analysis (see section 2): architectural style(s) used, domain quality model, main products' functionalities, products' semantics similarities in components and connectors (see Table 1).

Specification of products' architectures: Construct the architectural configurations of the products as a UML logic view from Table 1, see Figure 4. Dynamic aspects are not considered in this approach.

Table 1. Components and connectors of each product's architecture

OpenEMR	*PatientOS*	*Care2x*
a. Presentation Layer	a. Present. Layer	a. Present. Layer
Components:	Components:	Components:
a1. Web pages-Browser	-	a1
a2. Patient Portal	a2	a2
a3. Reports	a3	a3
-	a4. GUI	-
Connectores:	Connectors:	Connectors:
a1Ra2	-	a1Ra2
a1Ra3	-	a1Ra3
a1Rd1	-	a1Rd1
a1Rd2	-	a1Rd2
-	a4Ra2	-
-	a4Ra3	-
-	a4Rb5	-
b. Process Layer	b. Business logic	b. Process Layer
Components:	Components:	Components:
b1. Patient	b1. Patient business model	b1
b2. EHR Management	b2	b2
b3. Report System	b3	b3
-	b4. Mirth (HL7 engine)	-
-	b5. GUI – Server Side	-
Connectors:	Connectors:	Connectors:
b1Rc1	b1Rc1	b1Rc1
b1Rd1	X	b1Rd1
b1Rd2	-	b1Rd2
-	b1Rb5	-
b2Rc1	b2Rc1	b2Rc1
b2Rd1	X	b2Rd1
b2Rd2	-	b2Rd2
-	b2Rb4	-
-	b2Rb5	-
b3Rc1	b3Rc1	b3Rc1
b3Rd1	X	b3Rd1
b3Rd2	-	b3Rd2
-	b5Rb3	-
-	b5Rd1	-
c. Data Layer	c. Data Layer	c. Data Layer
Components:	Components:	Components:
c1. Data Base	c1	c1
c2. HL7 CDA Engine	-	c3. HXP Engine
Connectors:	Connectors:	Connectors:
c1Rc2	-	-
-	-	c1Rc3
d. Transmission	d. Transmission	d. Transmission
Components:	Components:	Components:
d1. Internet	d1	d1
d2. Satellite	-	d2

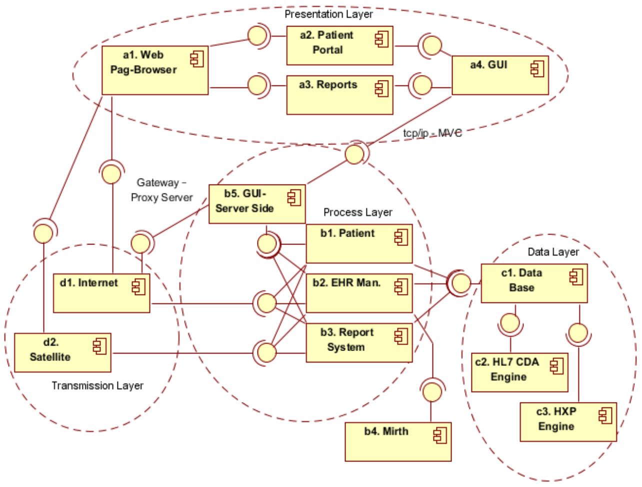
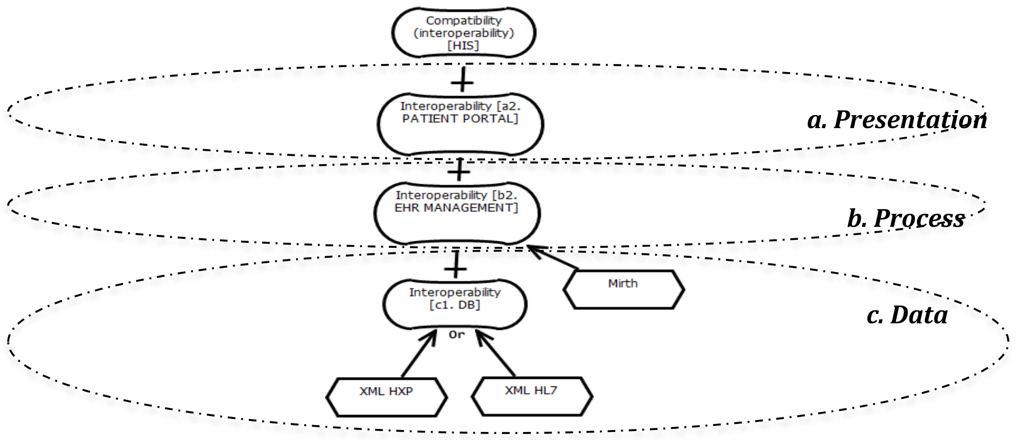


Figure 4. Initial Candidate Architecture automatically generated; the dashed oval dnotes the layer

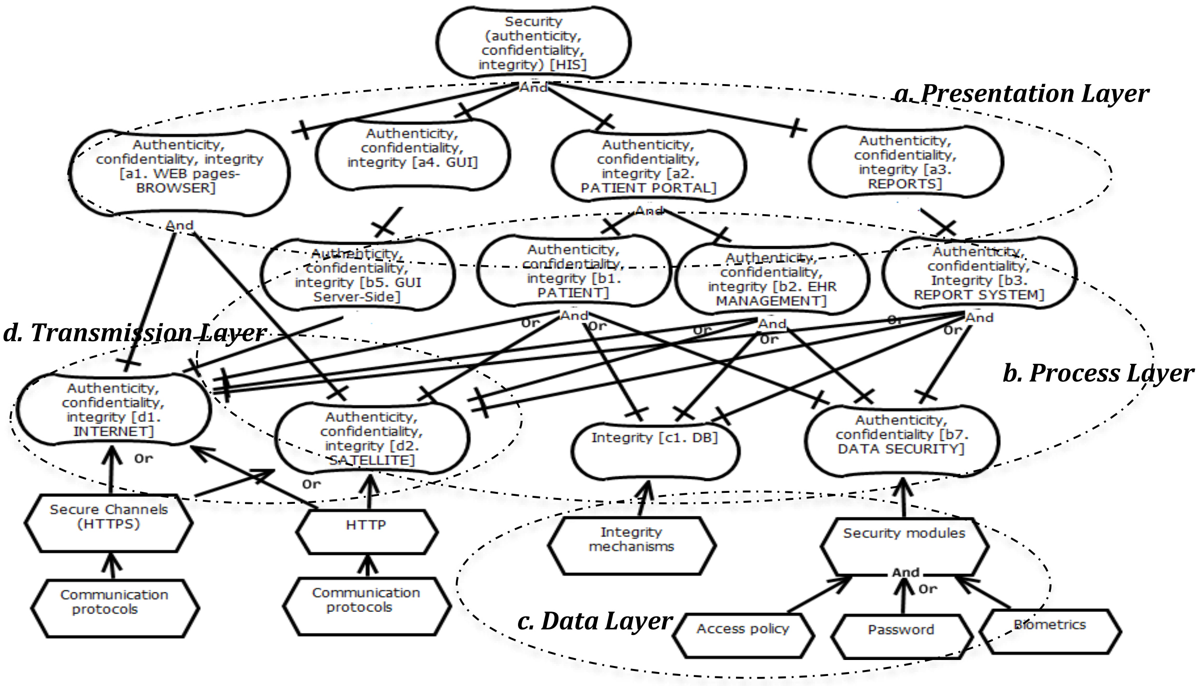
Construction of the Candidate Architecture: The initial *CA* is automatically produced by the *union* of the architectural configurations of each given product, i.e., the union of components of each products, preserving the connections. The *Common Core (CC)* is constituted by the common components; the others are considered *variant* *components*; variability of connectors is also identified. This step is performed automatically from the UML specifications of the architectural configurations of the products shown in Figure 4.

Completion of the Candidate Architecture: Construction of an *Extended Quality Model (EQM)* to offer solutions or scenarios for the quality properties necessary to accomplish functional or non functional requirements (see Table 2); *EQM* is the *domain quality model* defined in the Domain Analysis step, enriched by specifying each quality property and their architectural solutions realated to the *CA* components; it is used to construct a *SIG* model to identify the components related to non-functional requirements (see Figures 5 and 6). The completed *CA* configuration is expressed in UML (see and), signalling the variants components as stereotypes; connectors' variability can be observed in Table 1. This step is performed manually, however some of the activities can be authomatized, such as the *SIG* construction from the *EQM*.



Compatibility (interoperability) [HIS]
Interoperability [a2. PATIENT PORTAL]
a. Presentation
Interoperability [b2. EHR MANAGEMENT]
b. Process
Mirth
Interoperability [c1. DB]
c. Data
Or
XML HXP
XML HL7

Figure 5. *SIG* for *HIS EHR* interoperability in Presentation, Process and Data Layers



Security (authenticity, confidentiality, integrity) [HIS]
And
a. Presentation Layer
Authenticity, confidentiality, integrity [a1. WEB pages-BROWSER]
Authenticity, confidentiality, integrity [a4. GUI]
Authenticity, confidentiality, integrity [a2. PATIENT PORTAL]
Authenticity, confidentiality, integrity [a3. REPORTS]
And
And
Authenticity, confidentiality, integrity [b5. GUI Server-Side]
Authenticity, confidentiality, integrity [b1. PATIENT]
Authenticity, confidentiality, integrity [b2. EHR MANAGEMENT]
Authenticity, confidentiality, Integrity [b3. REPORT SYSTEM]
d. Transmission Layer
And
Or
Or
Or
And
Or
Or
And
Or
b. Process Layer
Authenticity, confidentiality, integrity [d1. INTERNET]
Authenticity, confidentiality, integrity [d2. SATELLITE]
Integrity [c1. DB]
Authenticity, confidentiality [b7. DATA SECURITY]
Or
Or
Secure Channels (HTTPS)
HTTP
Integrity mechanisms
Security modules
Communication protocols
Communication protocols
And
Or
c. Data Layer
Access policy
Password
Biometrics

Figure 6. *SIG* for *HIS* security

Construction of the Reference Architecture: *RA* is a valid architectural configuration, constructed from the competed *CA*, integrating variant components performing similar tasks into variation points; the variant connectors are defined between each pair of variation points, between a variation point and a common component or between two common components; constraints must be considered and new components can be introduced by the architect to obtain a more evolutive *RA* (see Table 3 and Figure 7). The *RA* configuration is expressed in UML and the components and connectors variation points are marked as stereotypes. A prototype computational tool is being designed to support the automation, whenever possible, of the proposed *RA* design process. In what follows the *RA* design refactoring process will be explained through the application to the *HIS* domain case study.

Table 2. Extended Quality Model (EQM)

Components of completed CA (Figure 8)	*Quality Model: quality characteristic(s)/sub-characteristic(s) with high 1, medium 2, low 3 priority*	*Architectural solution(s) or scenario(s) to accomplish quality characteristic(s) goal(s)*
a. Presentation Layer		
a1. Web pages - Browser	- Availability (1)	- d1. Internet and d2. Satellite networks via Web Server
	- authenticity (1), confidentiality (1), integrity (1) - time behaviour (2) - adaptability-scalability (2)	- HTTP/IP or secure channels SSL (HTTPS) - Communication protocols - Web services
a2. Patient Portal	- Availability-persistency (1), - authenticity (1), confidentiality (1), integrity (1), adaptability-scalability (2), precision (3)	- b1. Patient (subsystem in Process Layer)
	- Authenticity (1), confidentiality (1), integrity (1), availability-persistency (1), interoperability (1)	- b2. EHR Management (subsystem in Process Layer)
a3. Reports	- Authenticity (1), confidentiality (1), integrity (1), availability-persistency (1), precision (3)	- b3. Report System (subsystem in Process Layer) - b6. Algorithms (new component for precision in Process Layer) - b7. Data Security (new component for authenticlty and confidentiality in Process Layer)
a4. GUI	- Availability (1) - authenticity (1), confidentiality (1), integrity (1) - time behaviour (2) - modularity (3), modifiability (3)	- b5. GUI Server-Side for remote services; local services (cache), - Security components (new component b7) - Communication protocols (in the network) - MVC
b. Process Layer		
b1. Patient	- Precision (3), - integrity (1), adaptability-scalability (2), availability-persistency (1), - authenticity (1), confidentiality (1)	- Precision is solved by computation algorithms in b6 - c1. Data Base - b7. Data Security for authenticlty (biometrics, password) and confidentiality (access politics)
b2. EHR Management	- Integrity (1), adaptability-scalability (2), availability-persistency (1), interoperability (1) authenticity (1), confidentiality (1)	- c1. Data Base - Interoperability engine (solved in b4 or in c1, by c2 or c3) - b7. Data Security
b3. Report System	- Precision (3) - Availability-persistency (1), integrity (1), - authenticity (1), confidentiality (1)	- Precision is solved by computation algorithms in b6 - c1. Data Base - b7. Data Security
b4. Mirth Engine	- interoperability (1)	- XLM to HL7 engine
b5. GUI-Server Side	- modifiability (3), modularity (3) - availability (2), time behaviour (2), integrity (1)	- MVC - Façade - Proxy Server
b6. Algorithms	- authenticity(1), confidentiality (1) - Precision (3)	- b7. Data Security - Modules of computation algorithms
b7. Data Security	- authenticity(1) - confidentiality (1)	- Mechanisms to handle card, password, biometrics, etc. - Mechanisms to handle access rights policy
c. Data Layer		
c1. Data Base	- persistency (1), interoperability (1), scalability (2), adaptability (2), Integrity (1), availability (1)	- DBMS
	- interoperability (1)	- Mechanismos to translate XLM to HL7)
c2. HL7 CDA Engine		
c3. HXP Engine		- Mechanisms to translate HXP to HL7)
c4	- scalability (2)	- New Medical Standards in c4 (Mechanism to add new medical standards)
c5	- adaptability (2)	- JDBC for MySQL and PostgreSQL (Java platform)
c6	- adaptability (2)	- ODBC for MySQL and Oracle
c7	- persistency (1)	- Hibernate (Java platform)
c8	- integrity (1)	- Integrity Mechanisms specific to each database
c9	- availability (1)	- Availability mechanisms specific to each database
d. Transmission		
d1. Internet	- Time behaviour (2) - availability (2), adaptability-scalability (2) - integrity(1), authenticity(1), confidentiality (1)	- Communication protocols, Paging engine, - Web Server/Web services, Façade - Proxy server - HTTP, secure cannel SSL (HTTPS)
d2. Satellite	-Time behaviour (2) - availability (2), adaptability-scalability (2), integrity (1), authenticity(1), confidentiality (1)	- Communication protocols, Paging engine, - Web Server/Web services - HTTP, secure cannel SSL (HTTPS)

Table 3. Documentation of *HIS-RA* architectural elements

RA components/Connectors (Figure 7)	*CA Components/ Connectors (Figure 8)*	*Comments*
a. Presentation Layer		
Components		
<<a5>> User Interface	*a1. Web pages – Browser*	- User access control and interaction with HIS main functionalities and /or Web Server
	a4. GUI	- Client MVC-based GUI Note that a1 and a4 cannot be executed togetU
a2, a3	*a2. Patient Portal, a3. Reports*	- Common components to access main HIS functionalities
Connectors		
<<a5>>R<<d3>>	*{a1R d1, a1Rd2, a4Rb5Rd1};*	
<<a5>>Ra2	*{a1Ra2, a4Ra2};*	
<<a5>>Ra3	*{a1Ra3, a4Ra3};*	
b. Process Layer		
Components		
<<b11>> GUI Server	*b5. GUI Server-side*	- Access to functionalities in Process Layer and Internet remote services
<<b8>> Computation modules	*b6. Algorithms*	- Computations required for precise results
<<b9>> Security modules	*b7. Data Security*	- Services or algorithms to handle card, password, biometrics, etc. devices for user authentication and for policy of user access rights for confidentiality of medical information
b1, b2, b3	*b1. Patient, b2. EHR Manag., b3. Report System* *b4. Mirth Engine*	- Common components, main HIS functionalities
<<b10>>		
Connectors	*{b1Rd1, b1Rd2};*	- HL7 Engine for Interoperability
b1R<<d3>>	*{b2Rd1, b2Rd2};*	
b2R<<d3>>	*{b3Rd1, b3Rd2};*	- *b1Rd1, b2Rd1, b3Rd1* are variant connectors because they are absent in one of the preoducts
b3R<<d3>>	*{b3Rb6};*	
b3R<<b8>>	*{b1Rb6};*	
b1R<<b8>>	*{b1Rb7};*	
b1R<<b9>>	*{b2Rb7};*	
b2R<<b9>>	*{b3Rb7};*	
b3R<<b9>>	*{b2Rb4};*	
b2R<<b10>>	*b1Rc1; b2Rc1; b3Rc1;*	
b1Rc1; b2Rc1; b3Rc1;	*b1Rb5; b2Rb5; b3Rb5;*	
b1Rb5; b2Rb5; b3Rb5;	*{b5Ra4};*	
<<b11>>R<<a3>>	*{b5Rd1};*	
<<b11>>R<<d3>>		- a1 and a4 cannot be present at the same time - d2 cannot be used with b5
c. Data Layer		
Components		
<<c10>> Addition of new medical standards	*c4. New medical standards*	- Addition of new medical standards (catalogues for diagnosis, etc.) for system evolution
<<c11>> Data Integrity	*c8. Integrity mechanisms*	- Data integrity solutions, APIs specific to each DB
<<c12>> Data Availability	*c9. Availability mechanisms*	- Data availability mechanisms
<<c13>> Data Persistency	*c7. Hibernate*	- Data persistency mechanisms for Java platform
<<c14>> APIs	*c5. JDBC* *c6. ODBC*	- Portability mechanisms to other DBs under Java platform - Portability mechanisms to other DBs - c5 or c6 is mandatory
<<c15>> HL7 Data Model Engine	*c2. HL7 CDA Engine* *c3. HXP Engine*	- Interoperability with HL7 standard - c2 or c3 is mandatory
c1.	*c1. Data Base*	- Commercial Data Base; c1 is mandatory
Connectors		
c1R<<c10>>; c1R<<c11>>	*{c1Rc4}; {c1Rc8};*	
c1R<<c12>>; c1R<<c13>>	*{c1Rc9}; {c1Rc7};*	
c1R<<c14>>	*{c1Rc5, c1Rc6};*	
c1R<<c15>>	*{c1Rc2, c1Rc3}*	
d. Transmission Layer		
Components	*d1. Internet*	- Request/response for efficient, adecuate, available, secure and interoperable services using Internet protocols. Notice that d1 is a common mandatory component, however it has been included as variant since d2 is an optional variant and they share the same variation point <<d3>> Network. *d1* is mandatory, d2 is optional but it cannot be used with b5; HTTP or HTTPS protocols are optional for d1 and d2; HTTP is impossible without b7, due to HIS requirements
<<d3>> Network	*d2. Satellite*	

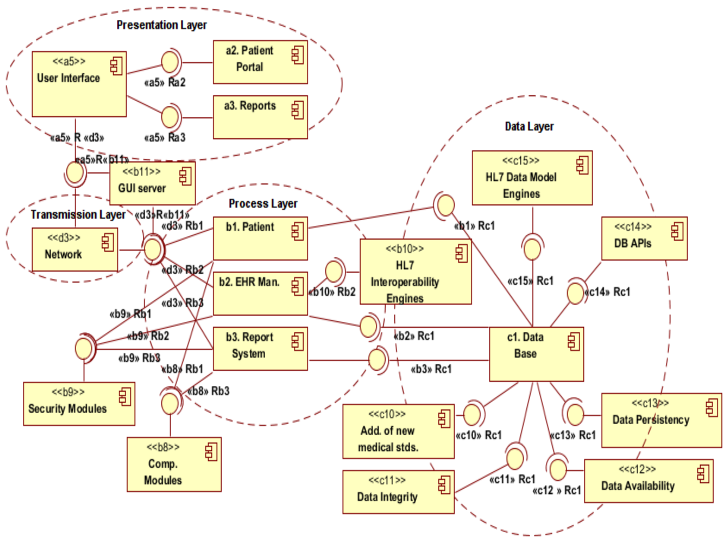


Figure 7. *HIS* Reference Architecture (*HIS-RA*)

3.2 Specification of the products' architecture

The brief Domain Analysis presented in section 2 has pointed out the main features of the domain relative to the architectural configuration that a family of systems in the domain should show. In our case, three *HIS* systems' architectures have been chosen as the input of our *RA* design refactoring process. Table 1 shows the components' names after having established a semantic similarity, i.e., components that are "similar" with respect to the functionality performed; if they have the same "functional" meaning or semantics, they have been given the same name. Components and connectors are grouped by layers; a connector is denoted by the relation *"R"* and the names of the connected components; Figure 3 shows the UML logic view of the three *HIS* architectures.

3.3 Construction of the Candidate Architecture

Similar products in the domain share globally the same architectural style(s) with a kernel of functionally similar components. Recall that the product architecture is represented by a connected graph or valid architectural configuration *(P,R)*, where *P* and *R* represent components and connectors of the product architecture. Given certain similar valid architectural configurations (P_i,R_i), $i = 1,2,3$, we have:

1. *CC* is the intersection of P_i of the given products' architectures; it is clear that since products are similar, i.e., they belongs to the same domain, *CC* is a non empty set (see Figure 5). In our case: *CC = {a2, a3, b1, b2, b3, c1, d1}*. Notice that components in *CC* will be in general mandatory components of every product generated from *RA*, since they are shared by existing products in the domain. Variant components are those not present in *CC*.

2. *Variability of connectors (VC1)* is conformed by those connectors that are not common to all the given products' architectures; they will be denoted by $\underline{xRy}$ indicating that components *x* and *y* are connected only in some architectures. In our case the connectors' variability is observed from Table 1, for example, connector *b1Rd1* is present in OpenEMR and Care2x and not in PatientOS, and it will be denoted by $\underline{b1Rd1}$, as a *variant connector*.

3. *Candidate Architecture (CA)* is a new valid configuration *(P,R)*, where *P* is the union of the sets of components in the configurations of each given product, and the connectors *R* are obtained according to the following definition: Let us denote by (P_i,R_i) $i=1,2,\ldots,s$ the valid architectural configuration of the *s* products considered, where P_i is the set of components or nodes and R_i is the set of edges or connectors. Let *CA=(P,R)* be the initial architecture obtained automatically, where $P=\cup P_i$ and $R=\cup R_i$, $i=1, 2,\ldots,s$. An *edge* $aRb \in R$ *is variant*, and will be denoted by $\underline{aRb}$ if and only If $\exists i, j$ such that $a, b \in P_i \cap P_j$ *and* $\exists aR_ib$ *and* $\neg\exists aR_jb$. The fact that $a, b \in P_i$ *and* $\neg\exists aR_ib$ *for some i*, was signalled in Table 1 by "X".

   From Table 1 and using the above definition, the following connectors are found: *a1Ra2, a1Ra3, a1Rd1, a1Rd2, a4Ra2, a4Ra3, a4Rb5, b1Rc1,* $\underline{b1Rd1}$, *b1Rd2, b1Rb5, b2Rc1,* $\underline{b2Rd1}$, *b2Rd2, b2Rb4, b2Rb5, b3Rc1,* $\underline{b3Rd1}$, *b3Rd2, b3Rb5, b5Rd1, c1Rc2, c1Rc3.*

4. *Variability of Components (VC2):* components of *CA* that are not present in *CC* are defined as *variant components*. This *CA configuration* automatically constructed constitutes an initial version of the reference architecture, because it already contains potential variants of components and connectors to construct the *RA* variability model. Figure 5 shows the UML logic view of *CA* produced by this union process of the three products' architectures.

The *SPL* variability is shown by the specificities of the products; in our case *VC2* is constituted by the non-common *CA* components, *VC2 = {a4, b4, b5, c2, c3}*. Namely, for PatientOs *a4. GUI* on the Presentation Layer, *b5. GUI Server-side* for modifiability, *b4. Mirth* for interoperability; *c2. HL7 CDA Engine* for OpenEMR and *c3. HXP Engine* for Care2X, also for interoperability in the Data Layer.

The variant connectors *VC1 = {b1Rd1, b2Rd1, b3Rd1},* were deduced from Table 1, and only one direction is considered; in certain cases the path used to connect a component may involve other components, such as *a4Rb5* to connect to Process Layer and *b5Rd1* to connect to Transmission Layer. Notice that a label is placed on certain connection interfaces to indicate the specificity of a certain product, for example the case of PatientOS that connects to the server-side by tcp/ip using MVC and establishes the internet connection through the Web Server with a gateway on the *b5. GUI Server-side*. Notice that this initial *CA* configuration contains all products' components and connectors, and each product could be reconstructed. *CA*, in our case, follows the 3-tiers style of the three products considered combined with the client-server/SOA model, including a Web Server (not shown in Figure 4) to access services to satisfy client requests, belonging to the *d. Transmission Layer* and this layer crosscuts all other layers, as it was shown in Figure 5: *a. Presentation Layer, b. Process Layer, c. Data Layer.* Two conflicting variants *a1* and *a4* for the user interface are present in this *CA* configuration; a constraint must be placed on the fact that both solutions cannot run in the Presentation Layer at the same time.

3.4 Completion of the Candidate Architecture

1. *Construction of the Extended Quality Model (EQM). EQM* is constructed in Table 2, from the domain quality model discussed in section 2, considering the possible scenarios or choices of solving (provide an architectural solution or mechanism) quality requirements associated to each *CA* component; the component, the set of quality characteristics related to the component with an assigned priority ranked 1 high, 2 medium, 3 low, respectively, and the possible architectural solutions for each quality characteristic, are shown. Quality attributes and metrics can also be included in *EQM*, if a lower granularity is required [10, 48] (see the Appendix for general metrics). Notice that the knowledge provided by *EQM* is captured from the Domain Analysis, from the documentation of each product and the architect's experience.

2. *Construction of the SIG. EQM* is used to build the *SIG* [14, 15], which is a graph of high abstraction level of non-functional requirements (*softgoals*), which are refined into architectural solutions (*operationalizations*); these are then associated to architectural components; the *"OR"* clause, placed as labels on the arcs indicates optional decomposition; the *"AND"* clause, also a label on the arcs, indicates that there is no alternative (mandatory); hence *OR* can indicate variability; a *SIG* node is composed by a list of quality characteristics or properties denoting quality requirements, and their associated components, according to Supakkul and Chung [15]: *type* (list of softgoals; in ISO/IEC 25010 terminology quality characteristics represent the quality model or non-functional goals to be accomplished by the component) and [*name*] the *topic* or context of action represents the component(s) affected by the listed quality characteristics; many graphical notations have been proposed for the *SIG*, however they share the fact of being complex and non-standard; we will use here the GRL (Goal Requirements Language) terminology [16], where the crossed line between two components or topics (rounded shape), indicates "decomposition" or refinement and the directed arrow from the operationalization (indicated by an exagonal shape) to the softgoal, indicates a "contribution" towards the solution of the softgoal. Operationalizations correspond directly to architectural solutions or mechanisms; The layers have been marked on the *SIG* as ovals with dashed lines. We have chosen to show only some relevant softgoals (interoperability and security) related to certain contexts of action or topics (see Figures 6 and 7).

   SIG helps to identify by refinement, new components required to accomplish specific quality requirements which will be included in *RA* as new variation points; in our case, it is constructed using *EQM* (see Table 2) which contains the *HIS* domain quality model specification, profiting of the ISO/IEC 25010 [11] hierarchical decomposition of quality characteristics (see Figure 2). Softgoals attached to each component, specified by this quality model, are refined into operationalizations, which are represented by low-level components that can be variants to be "customized" later on into the different products of the *SPL*. Figure 6 shows the *SIG* for the *interoperability* quality property and the traceability of this property to each component involved is clearly established. In Figure 7, security softgoals *authenticity* and *confidentiality* related to component [b7. Data Security], which a new component operationalized into *"Security modules"* that it is further refined into particular solutions or mechanisms to handle authentication, namely *"Password"* or *"Biometrics"*, and *"Access policy"* for confidentiality, to manage the user's access rights. Since the Transmission Layer crosscuts all layers due to the SOA style, *security* related to message transmission is also solved by *d1. Internet* and *d2. Satellite* communication protocols involving "Secure channels" SSL (HTTPS) and "HTTP" operationalization components. *EQM* is then completed (see Table 2) with the new components, which are variants introduced during the SIG construction to satisfy quality goals and enrich the architecture flexibility.

3. *Assessment of contributions.* Priorities assigned to each quality property (see Table 2), help to handle the *SIG* complexity since it can be constructed starting with the quality property with greatest priority, and so on; priorities also help to establish tradeoffs between quality properties when contributions are defined, for example, in the *HIS* case, in *b2. EHR management*, the quality properties *security (authenticity and confidentiality)* with priority 1 and *adaptability-scalability* with priority 2 (see Table 2); clearly the system must satisfy security over adaptability for the *EHR* component; hence a security component must be present in *RA*. We have not placed explicit softgoals contributions on the *SIG* (usually denoted by +, -, ++, --

) for this case study, to ease the figures' legibility, however it can be appreciated that *efficiency* (time behaviour) of the Web-pages interface can be affected positively or negatively, depending on the paging engine of the Web Server, and it can also affect negatively the *availability*. On the other hand, availability can be improved with the separated GUI; the *efficiency*, however, can be affected negatively, and *modifiability* is greatly favoured by the separation of concerns due to the MVC pattern and not from the Web services adaptability-scalability claimed by *SOA*. The separated *GUI* also favours the system modularity since the Gateway or Proxy Server is accessed through the *Façade* pattern. A separate table showing positive and negative contributions of quality characteristics could be included [10, 48], or they could be added to Table 2.

4. *Variability modelling. SIG* helps also to model variability, as we have noticed from Figures 5 and 6. It helps to identify new component variants that can appear during the softgoals refinement process; the quality properties can be completely traced and evaluated on the *SIG*, to help corroborate or justify the architectural choices/solutions extracted from the products' documentation or added by the architect expertise (see Table 2). Since the *AND/OR* arcs indicate manadatory or optional issues, they are used to model component variability, in the FODA (Feature-Oriented Domain Analysis) way [19], and also to detect inconsistency in the sense that, certain components (for example a1 and a4 for the user interface) cannot be present at the same time in a product configuration. However, variability of connectors is not treated in FODA. It is clear that the *SIG* construction process is complex, since it has to be done for each component and each quality property; moreover, *SIG* is not expressed by a standard notation, being these some of the major limitations of the *SIG* approach; however the *SIG* construcion can be managed taking into account information in Table 2 on the priorities of the quality properties, and also using available automatic tools [51]. As we have already pointed out, layers have been marked on the *SIG* as ovals with dashed lines. After analysing the *AND/OR* decomposition on the *SIG*, the final UML logic view of the completed *CA* architecture is produced, signalling the variant components with the *<<variant>>* stereotype (see Figure 8).

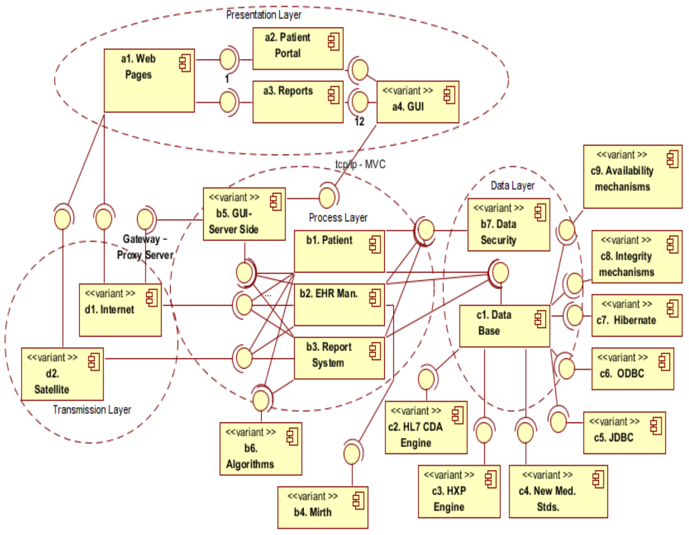

Presentation Layer
a1. Web Pages
a2. Patient Portal
a3. Reports
<<variant>> a4. GUI
1
12
tcp/ip - MVC
<<variant>> b5. GUI-Server Side
Gateway - Proxy Server
Process Layer
b1. Patient
b2. EHR Man.
b3. Report System
<<variant>> b6. Algorithms
<<variant>> b4. Mirth
<<variant>> d1. Internet
<<variant>> d2. Satellite
Transmission Layer
Data Layer
<<variant>> b7. Data Security
c1. Data Base
<<variant>> c9. Availability mechanisms
<<variant>> c8. Integrity mechanisms
<<variant>> c7. Hibernate
<<variant>> c6. ODBC
<<variant>> c5. JDBC
<<variant>> c2. HL7 CDA Engine
<<variant>> c3. HXP Engine
<<variant>> c4. New Med. Stds.

Figure 8. The completed *HIS* Candidate Architecture (*HIS-CA*)

In the Presentation Layer, *a1* y *a4* are two variants to construct the user interface: *a1. Web pages-Browser* requires a Web Server with a paging engine, or a separated *a4. GUI* with local services; it communicates by RPC and tcp/ip with the Pocess Layer and uses the MVC pattern for separation of concerns to ensure modifiability-scalability; it executes remote services in the Process Layer through a Gateway or Proxy Server (*b5. GUI-Server Side*), connecting with a Web Server. In the Process Layer, the new component *b6. Algorithms* is introduced as a variant to get precision in the algorithmic processes requiring it and *b7. Data Security* is a new variant component introduced to satisfy the authenticity, and confidentiality softgoals [36] respectively. However, some of the systems are limited to satisfy security requirements in the Transmission Layer using only network protocols.

In the Data Layer, for interoperability we have *c2. HL7 CDA* and *c3. HXP Engines*; *c4. New medical standards* [36] is a component included to ensure scalability for evolution with the addition of new medical standards; the portability (adaptability) to different databases depends on the APIs variants *c5. JDBC (MySQL* and *Oracle)* for Java platform*, c6. ODBC (MySQL* and *PostgreSQL)* used; the *c7. Hibernate* system is a solution for persistency also for Java platform; *c8. Integrity mechanisms* should include available solutions for data integrity at database level, and finally *c9. Availability mechanisms* should include available solutions for data availability, such as mirror databases etc. Component *b4. Mirth* engine, is another option for interoperability, with an HL7 messaging engine running in the Process Layer to ensure also the system capacity of evolution or maintainability.

With respect to the transmission, two alternatives are found: *d1. Internet, d2. Satellite* (network for mobile devices). *SOA* claims portability (adaptability-scalability) through the use of Web services, with respect to *d1* and *d2*, including the Web Server (for example Apache) and network security protocols. Notice that in the completed *CA* new variant components with their connectors have been added in the *SIG* refinement process, to satisfy quality requierements, namely *b6, b7, c4, c5, c6, c7, c8* and *c9*; the connectors of these new components are: *b6Rb1, b6Rb3, b7Rb1, b7Rb2, b7Rb3, c4Rc1, c5Rc1, c6Rc1, c7Rc1, c8Rc1, c9Rc1*. This issue is not treated in most of the FODA-based approaches, where only component variability is focused. All the variants of the completed *CA* will be generalized (grouped according to similar tasks) into *RA* variation points, representing abstract architectural elements (components or connectors) to obtain *RA* as the final artefact.

3.5 Construction of the Reference Architecture

This is also a manual process based on the completed *CA* configuration (see Figure 8) and on the architect expertise. The variants of *HIS-CA* are studied by the architect to be generalized into into abstract instantiable components, the variation points; however, since our process is bottom-up, all variants present in *CA* are extracted from the products; in order to make better choices, the architect could provide other alternatives or scenarios that could be added to Table 2, to enrich the variants obtained and discuss more tradeoffs. *RA* valid architectural configuration is constituted by two types of nodes:

- the nodes conformed by grouping the components of *CA* with similar tasks, and will be denoted by the stereotype *<<component name>>*;
- common *CC* components of *CA* directly denoted by their names.

The *HIS-RA* nodes (see Figure 7) are the following: *<<a5>>={a1, a4}; a2; a3; <<d3>>={d1, d2}; b1; b2; b3; <<b8>>={b6}; <<b9>>={b7}; <<b10>>={b4}; <<b11>>={b5}; c1; <<c10>>={c4}; <<c11>>={c8}; <<c12>>={c9}; <<c13>>={c7}; <<c14>> = {c5, c6}; <<c15>>={c2, c3}.*

The connectors between each pair of nodes are defined considering the relation between the *CA* components conforming the given nodes. Given two nodes *<<x>>, <<y>>* we denote *<<x>>R<<y>> = {aRb:a∊ <<x>>, b∊<<y>>}* and given two nodes *x* and *<<y>>* we have *xR<<y>> = {xRb:b∊<<y>>}*. The *HIS-RA* connectors (see Figure 7) will be the following:

<<a5>>Ra2={a1, a4}Ra2={a1Ra2, a4Ra2};
<<a5>>Ra3={a1, a4}Ra3 ={a1Ra3, a4Ra3};
<<a5>>R<<d3>>={a1, a4}R{d1, d2}={a1Rd1, a1Rd2, a4Rb5Rd1};
b1R<<d3>>=b1R{d1, d2}={b1Rd1, b1Rd2};
b2R<<d3>>=b2R{d1, d2}={b2Rd1, b2Rd2};
b3R<<d3>>=b3R{d1, d2}={b3Rd1, b3Rd2};
b3R<<b8>>={b3Rb6};
b1R<<b8>>={b1Rb6};
b1R<<b9>>={b1Rb7};
b2R<<b9>>={b2Rb7};
b3R<<b9>>={b3Rb7};
b2R<<b10>>={b2Rb4};
b1Rc1; b2Rc1; b3Rc1; b1Rb5; b2Rb5; b3Rb5;
<<b11>>R<<a3>>={b5}R{a1,a4}={b5Ra1, b5Ra4}={b5Ra4};
<<b11>>R<<d3>>={b5}R{d1,d2}={b5Rd1}, b5Rd2}={b5Rd1};
c1R<<c14>>=c1R{c5, c6}={c1Rc5, c1Rc6};
c1R<<c10>>={c1Rc4};
c1R<<c11>>={c1Rc8};
c1R<<c12>>={c1Rc9};
c1R<<c13>>={c1Rc7};
c1R<<c15>>=c1R{c2, c3}={c1Rc2, c1Rc3}.

Table 3 shows the list of the *HIS-RA* common components, variation points and their corresponding connectors; the *HIS-RA* variation points are: *<<a5>> User interface,* in Presentation Layer; in Transmission Layer *<<d3>> Network*, includes the Web Server connections and SSL with specific network security protocols, not shown in Figure 7; in Process Layer *<<b10>> HL7 Interoperability Engines* indicates commercial available mechanisms for interoperability; *<<b11>> GUI Server* includes *b5. GUI Server-side* variant (considered in *<<a5>>* as the *a4. GUI Client-side* variant); in Data Layer *<<c15>> HL7 Data Model Engines* indicates different mechanisms for *HL7 CDA* or similar models; other variation points are *<<b8>> Computation modules* to handle computation algorithm for precise results from Patient and Reports, and *<<b9>> Security modules* to handle authenticity and confidentiality issues, as it was explained before. The Data Layer contains variation points *<<c14>> DB APIs,* for portability, used to handle different kinds of commercial databases *(MySQL, PostgreSQL, Oracle), <<c10>> Addition of new medical standards I* for evolution, and the variation points *<<c11>> Data integrity, <<c12>> Data Availability,* and *<<c13>> Data Persistency*; Figure 7 shows all the *HIS-RA* variation points, variant connectors and common components.

4. RA instantiation to derive new products' architectures

RA variation points and their respective variant connectors are instanciated to generate new products. The specific product architecture derivation process consists basically in making a choice among a huge number of possible elements of variation points and variant connectors. The elements or instances of the set of variant connectors have to be chosen carefully to perform the instantiation, and many of these selections could result in non-valid architectural configurations. Figure 9 shows an example of how *HIS-RA* can be used to generate the architecture of a new *HIS*.

We have to recall that common components in the common core are mandatory, hence they have to be included in the product that is going to generated. In general, to perform the *RA* instantiation, the functional and non-functional requirements for the particular *SPL* product requested by a client, are needed as a main input. Constraints on components and/or connectors have to be considered (see Table 3). Notice that the architecture generated from *HIS-RA* for the new product must be a valid configuration, i.e. all components must be connected and connectors' instances must be non-empty. It is clear that additional architectural solutions or mechanisms can be included, trusting the architect's experience and technology evolution.

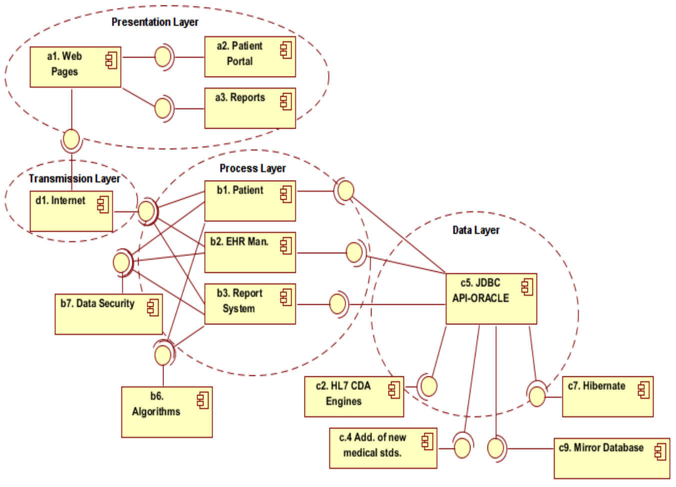


Figure 9. Product derivation with *HIS-RA*

The *HIS* product shown in Figure 9 has the *a4. Web-pages* interface to access, via a2. Patient Portal and *a3. Reports*, the basic functionalities for patient attention *b1. Patient, b2. EHR Management* and *b3. Reports* for statistical, medical and administrative purposes; a special b7. Data Security component is located in the Process Layer besides the message security provided by standard Internet protocols (d1); the b6. Algorithms component is used to get the required precision of computations; in the Data Layer we have *c5. ORACLE*, Java portable with *API JDBC*; interoperability is assured with *c2. HL7 CDA* Engine, and evolution is attained by *c4. Addition of new medical standards*; data persistency to Java platform is solved with *c7. Hybernate*, and finally *c9. Mirror database* is kept for data availability. It can be observed that all the priority quality requirements have been properly solved (interoperability, security, availability, persistency andadaptability-scalability); the Internet connection is the responsible for time-behaviour (efficiency) and can also affect system and services availability.

5. Related works

5.1 RA design for SPL

In general, the design of *RA* for the *SPL* context appears to be a complex process because the artifacts produced must be evolutionary and reusable. Reengineering techniques have been used to modify an existing *RA* that in general has been built within a proactive top-down approach using domain knowledge; from this generalization, new evolutionary *RAs* are built in [9, 20, 21, 37, 38]. Architectural design methods that have been used to develop the architecture of single systems have been adapted to the *RA* design for *SPL*, such as Lamswerdee, Chung et al. and often combined with FODA [19]; component-oriented architectural design method, such as Weiss and ADD (Attribute Driven Design) [30] have also been modified for *SPL* and the UP (Unified Process) software development process is centered in constructing a baseline architecture, on top of which the whole software system has to be incrementally built; in particular ADD uses quality scenarios containing architectural drivers, which is a notion similar to our qulity model, to relate architectural solutions (attribute primitives) to quality requirements, as we do in *EQM*; however they do no relate them to components at this stage as we do in *EQM* and they do not use the *SIG*. Bosch's method instead was originally defined in a *SPL* context. A more detailed discussion on these approaches can be seen in [36]. However, very few of them consider standard specification of non-functional requirements and in the literature very few guidelines provide the design process details; the lack of standard notations to specify processes and architectures is also observed. Common points are that non-functional requirements drive the whole architectural design and that the architect expertise play a very important role and cannot be avoided.

5.2 Refactoring techniques

Refactoring techniques, used in bottom-up design, have been traditionally used to reconstruct legacy code and reverse engineering has been used to recover or reconstruct product documentation [39]. The works of [40, 41] have similarity with our approach, however none of them use a graph model to represent architectural configurations, nor the ISO/IEC 25010 standard is used to specify non-functional requirements as quality requirements.

5.3 Variability modelling

Many works are found in the literature in the context of variability modelling, on functional requirements variability with FODA-based feature models, which is an important aspect considered in the *RA* design, to derive new products for the *SPL*. However, the variability modelling of non-functional requirements has not been worked as much [42, 49]; several works use the *SIG* approach in combination with the FODA feature model, namely, F-SIG [18] and EFM (Extended Feature Model) [43]; we found only one method proposed in [17] that uses

EFM, F-SIG and the specification of quality requirements by ISO/IEC 9126-1, which is the old version of ISO/IEC 25010; however, they do not model connectors' variability, and as we have already pointed out, they follow a top-down approach and construct the feature model; in our bottom-up approach, features are captured from the refactoring of existing products and we do not need to construct the feature model. The process defined in [10, 48] inspires this work, however variability modelling is not treated and the automatic *CA* construction is not properly justified; moreover, the *SIG* approach in combination with the standard ISO/IEC 25010 specification is introduced in our new process to extend the variability model to non-functional issues considering a standard specification of quality requirements (see Appendix).

5.4 *HIS* Reference Architecture

In the context of *SPL* and *HIS-RA* to handle *EHR* for interoperability, work has been going on, considering *SOA* for data exchange and communication [36, 44], and focusing on model-driven architecture applied to the HL7 CDA standard [45]. In [25] a *HIS* design process is proposed, not directly related to *SPL*; the approach is not specifically *HIS*-driven and could be applied to any domain. In view of the variety of *HIS* products, the absence of a commonly adopted *HIS-RA*, and the proliferation of medical standards, not fully adopted or approved, we propose the semi-automatic bottom-up approach based on the refactoring of existing products, described in this paper.

6. Conclusions

A general *RA* design based on a refactoring process has been defined and applied to the *HIS* domain. The importance of designing a *HIS-RA* or generic architecture satisfying the domain priority quality requirements demanded by main functionalities that must be present in a modern *HIS*, namely security, persistency, interoperability and availability, has been focused. Three widely used open-source *HIS* have been studied to analyse similarity and their architectures refactored, as the input to a semi-automatic bottom-up process to construct the *HIS-RA*. A Candidate Architecture *(CA)* is obtained automatically as a first version of the *HIS-RA*, to identify common and variant components and connectors; *CA* it is then completed to enrich the variability modelling with non-functional issues using the *SIG* model.

Our approach is general and can be applied to any domain using SOA/Layers architectural styles. The *CA* completion and the *RA* construction steps can be partially authomatized; in case of a great number of products, the number of components and the variant elements of non-functioanl nature may increase dramatically, and the handling of combinatorial complexity by an automatic process is a great advantage. However, as in all architectural design methods, the architect's experience remains irreplaceable and plays an important role in determining the "best" architectural solution w.r.t. a set of non-functional requirements. In our context, *HIS-RA* is conformed by the SOA/Layers styles used in the domain with the core of main domain functionalities and components derived from the satisfaction of domain non-functional requirements, specified by the ISO/IEC 25010 quality model [11]. As we have already pointed out, non-functional variability modelling has been integrated into the refactoring process using the *SIG* model [14, 15], to facilitate variants' detection and complete the initial architectural *CA* configuration obtained automatically with new components introduced to satisfy non-functional requirements. The derivation of the architecture of a concrete product from *RA* has been presented as an example of the *RA* usefulness; a process for products derivation from *RA*, inspired in [49] is an on-going work, and a computational tool to support the complete bottom-up refactoring process is under design.

7. Aknowledgment

This work has received partial support from different national projects and institutions: projects PEII DISoft 2011001343 of Fonacit, Venezuela and DARGRAF PG 03-8730-2013-1 of the CDCH (Consejo de Desarrollo Científico y Humanístico), Venezuela Central University, the Postgraduate Studies of Computer Science, Faculty of Science, Venezuela Central University, and the Banco Cemtral de Venezuela.

References

[1] Clements, P. & Northrop, L. (2001). Software product lines: practices and patterns. Readings: Addison Wesley.
[2] Pohl, K., Böckle, G. & van der Linden, F. (2005). Software product line engineering - Foundations, principles, and techniques. USA: Springer.
[3] Nakagawa, E., Antonio, P. & Becker, M. (2011). Reference architecture and product line architecture: A subtle but critical difference. Lecture Notes in Computer Science 6903, pp. 207-211.
[4] Shaw, M. & Garlan, D. (1996). Software Architecture: Perspectives of an emerging discipline. New York: Prentice-Hall.
[5] Clements, P. & Krueger, C. (2002). Point-Counterpoint: Being proactive pays off - Eliminating the adoption. IEEE Software 19(4), pp. 28-31.
[6] Rashid, A., Royer, J. & Rummler, A. (2011). Aspect-oriented model-driven software product lines: The AMPLE way. Cambridge: Cambridge University Press.
[7] Matinlassi, M. (2004). Comparison of software product line architecture design Methods: COPA, FAST, FORM, KobrA and QADA, Proceddings 26th International Conference on Software Engineering (pp. 127-136). May 23-28. Edinburgh. Scotland.
[8] Chikofsky, E. & Cross, J. (1990). Reverse engineering and design recovery: A taxonomy. IEEE Software 7(1), pp. 13-17.
[9] Rashid, A., Royer, J. & Rummler, A. (2011). Aspect-Oriented Model-Driven Software Product Lines. The AMPLE Way. Chap. 2, pp. 48 & Chap. 5, pp. 125-157. Cambridge: Cambridge University Press.
[10] Losavio, F. et al. (2013). Graph modelling of a refactoring process for product line architecture design. XXXIX Latin American Computing Conference (pp. 1-12). October 7-11. Naiguata, Venezuela.
[11] ISO/IEC (2011). ISO/IEC JTC1/SC7/WG6. Software and Systems Engineering. USA: IEEE.
[12] Krutchen, P. (1995). The 4+1 view model of software architecture. IEEE Software 12(6), pp. 42-50.

[13] Object Management Group. (2005). Unified Modelling Language Superstructure, version 2.0. USA: OMG.
[14] Chung, L. et al. (2000). Non-functional Requirements in Software Engineering. Massachusetts: Springer.
[15] Supakkul, S. & Chung, L. (2004) Integrating FRs and NFRs: A use case and goal driven approach. Proceedings 2nd International Conference on Software Engineering Research, Management, and Applications (pp. 30-37). May 5-7. Los Angeles, USA.
[16] Amyot, D. et al. (2009). A lightweight GRL profile for i* modeling. Lecture Notes in Computer Science 5833, pp. 254-264
[17] Gurses, O. (2010). Non-functional variability management by complementary quality modeling in a software product line. MSc thesis, Graduate School of Natural and Applied Science of Middle East Technical University. Turkish.
[18] Jarzabek, S., Yang, B. & Yoeun, S. (2006). Addressing quality attributes in domain analysis for product lines. IEEE Proceedings Software 153(2), pp. 61-736.
[19] Lee, K., Kang, K. & Lee, J. (2002). Concepts and guidelines of feature modeling for product line Software Engineering. Lecture Notes in Computer Science 2319, pp. 62-77.
[20] Westfechtel, B. & von der Hoek, A. (2003). Software architecture and software configuration management. Lecture Notes in Computer Science 2649, pp. 24-39
[21] Bosch J. (2000). Design and use of software architectures - Adopting and evolving a product-line approach. USA: Addison-Wesley.
[22] De la Torre, I. et al. (2010). Categorización de los estándares de las HCE. España: Universidad de Valladolid.
[23] Monteagudo, J. & Hernández, S. (2002). Estándares para las Historias Clínicas Electrónicas. Online [January 2015].
[24] Open Clinical (2011). Electronic medical records, Electronic Health Records. Online [January 2015].
[25] Pazos, P. (2010). Aplicación de estándares en la HCE. Proceedings 39 Jornadas Argentinas de Informática. Agosto 30 – Septiembre 3. Buenos Aires, Argentina.
[26] Pazos, P. (2010). Taller de OpenEHR: Un Estandar para crear HCEs. Proceedings 39 Jornadas Argentinas de Informática. Agosto 30 – Septiembre 3. Buenos Aires, Argentina|.
[27] OpenEHR Foundation. http://www.openehr.org
[28] Holzinger, A., Burgsteiner, H. & Maresch, H. (2009) Experiences with the practical use of Care2x in medical informatics education. Online [Nov. 2014].
[29] Samilovich, S. (2010). OpenEMR. Online [Oct. 2014].
[30] Bass, L., Klein, M. & Bachmann, F. (2001). Quality attribute design primitives and the attribute driven design method. Lecture Notes in Computer Science 2290, pp. 169-186.
[31] http://www.medfloss.org/node/161.
[32] http://cosgov.ow2.org/xwiki/bin/download/Main/Sessions/hart.pdf.
[33] http://www.care2x.org
[34] World Wide Web Consortium. (2004). Web Services Architecture Requirements. Online [Sep. 2014].
[35] Latorrilla, E. (2004). HXP Healthcare eXchange Protocol. Online [Nov. 2014].
[36] Levy, N., Losavio, F. & Pollet, Y. (2014). Architecture et qualité de systèmes logiciels. In Oussalah, M. (Ed.), Architectures logicielles: Principes, techniques et outils. Chap. 7. Paris: Hermes Science Publications
[37] Critchlow, M. et al. (2003). Refactoring product line architectures. In IWR: Achievements, Challenges, and Effects, pp. 23-26.
[38] Saraiva, D. et al. (2010). Architecting a model-driven aspect-oriented product line for a digital TV middleware: A refactoring experience. Lecture Notes in Computer Science 6285, pp. 166-181.
[39] Fowler, M. (1999). Refactoring. Improving the design of existing code. USA: Addison-Wesley.
[40] Koziolek, H., Weiss, R. & Doppelhamer, J. (2009). Evolving industrial software architectures into a software product line: A case study. Lecture Notes in Computer Science 5581, pp. 177-193.
[41] Wu, Y. et al. (2011). Recovering object-oriented framework for software product line reengineering. Lecture Notes in Computer Science 6727, pp. 119-134.
[42] Perrouin, G. (2011). A metamodel-based classification of variability modeling and software product lines. Online [Nov. 2014].
[43] Benavides, D., Segura, S. & Ruiz, A. (2010). Automated analysis of feature models 20 years later: A literature review. Information Systems 35(6), pp. 615-636.
[44] Cohen, S. (2009). SOA in electronic health record product line. Carnegie Mellon University.
[45] Raka, D. (2010). A product line and model driven approach for interoperable EMR messages generation. Master Report, California State University. USA.
[46] http://www.hl7latam.org/
[47] IEEE. (1998). IEEE-std-1061-1998. Standard for a Software Quality Methodology. USA: IEEE Computer Society.
[48] Losavio, F., Ordaz O. & Levy, N. (2014). Refactoring Graph for Reference Architecture Desogn Process. Proceedings Approches Formelles dans l'Assitence au Développement de Logiciels (pp.103-108). June 11-12. Paris, France.
[49] Siegmund, N. et al. (2012). SPL Conqueror: Towords optimization of non-functional properties in software product line. Software Quality Journal 20(3-4), pp. 487-517.
[50] Karopka, T., Schmuhl, H. & Demski, H. (2014). Free/Libre Open Source Software in Health Care: A Review. Healthcare Information Research 20(1), pp. 11–22.
[51] http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome. [Nov. 2014].

Appendix
HIS Domain Quality model (HIS-DQM)

Quality characteristic for HIS domain with priority high 1, medium 2, low 3	*ISO/IEC 25010 definition of quality characteristic/sub-characteristic*	*General Metrics*
1 Compatibility	The degree to which two or more systems or components can exchange information and/or perform their required functions while sharing the same hardware or software environment	*Interoperability* X=A/B, A=Number of interface data formats that have been implemented correctly as in the specifications
- interoperability of web services and of electronic medical data	- the degree to which two or more systems or components can exchange information and use the information that has been exchanged	B=Number of data formats to be exchanged as in the specifications, 0≤X≤1, the closer to 1 the better
1 Security	The degree to which information and data are protected so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them	*Data corruption prevention* X=A/B, A= Number of implemented instances of data corruption prevention as specified confirmed in review. B= Number of instances
- authenticity	- the degree to which the identification of a subject or resource can be proved to be the one claimed	of operation/access identified in requirements as capable of corrupting/destroying data ,
- confidentiality	- the degree to which information and data are protected from unauthorized disclosure of information or data, whether accidental or deliberate	0≤X≤1, the closer to 1 the better *integrity:* Provided by HTTP, SSL (HTTPS) Network transport protocols at transmission
- integrity for user authentication and policies to access patient and medical data	- the degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data	*Availability:* Total time during which the product or component is in "up" state;
1 Reliability *- availability- persistency*	The degree to which a system or component performs specified functions under specified conditions for a specified period of time.	*Adaptability-scalability:* X=A/B, A = No of functions capable of achieving desired results in specific context
2 Portability *- adaptability-scalability* for new and increasing volume of data	- the degree to which a system or component is operational and accessible when required for use; availability is therefore a combination of maturity (which governs the frequency of failure), fault tolerance and recoverability (which governs the length of down time following each failure).	B = Total No of functions, 0≤X≤1, the closer to 1 the better *Modularty:* Cyclomatic complexity
3 Maintainability	- the degree to which a system or component can be effectively and efficiently transferred from one hardware,	
- modularity	software or other operational or usage environment to another	*Modifiability:* X=A/B, A= No of confirmed changes, B = Total No of changes, 0≤X≤1; the closer to 1 the
- modifiability or system evolution	- the degree to which the product can effectively and efficiently adapted for different specified hardware, software or other operational or usage environments; it includes the scalability of internal capacity	better; it can be estimated considering the No of LOC added *Time behaviour:* Depends on the programming language and processor used; is usually
2 Efficiency (Performance)	- The degree of effectiveness and efficiency with which the product can be modified	measured at run-time. It can be estimated as the total time spent to accomplish a
- time behaviour such as response-time, latency and throughput	- the degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components	functionality; is usually measured at run-time *Resource utilization:* LOC (lines of code), memory (size in bytes)
- resource utilization such as memory, energy, LOC	- the degree to which a product can be effectively and efficiently modified without introducing defects or degrading performance	and power used; are usually measured at run-time;
3 Functional suitability *– completeness* w.r.t. commonality	- The performance relative to the amount of resources used under stated conditions	*Completeness:* X=1-A/B, A=Number of missing functions detected in evaluation. B=Number of functions described in requirement specifications; [0≤w≤1], the closer to 1, the more complete.
- correctness- precision for computations	- the response and processing times and throughput rates of a system when performing its function, under stated conditions in relation to an established benchmark - the amounts and types of resources used when the product performs its function under stated conditions in relation to an established benchmark - the degree to which the product provides functions that meet stated and implied needs when the product is used under specified conditions - the degree to which the set of functions covers all the specified tasks and user objectives - the degree to which the product provides the correct results with the needed degree of precision	*Precision:* X=A/B, A= Number of data items implemented with specific levels of precision, confirmed in evaluation; B= Number of data items that require specific levels of precision, 0≤X≤, the closer to 1 the better