Revista Venezolana de Computación
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 4, No. 1, pp. 10-21, Junio 2017
Fecha recepción: 29/04/2017, Fecha aceptación: 30/07/2017

Experience on Software Product Line Domain Engineering for Mobile Computing

Francisca Losavio[1], Oscar Ordaz[1,2]
francislosavio@gmail.com, oscarordaz55@gmail.com

[1] Escuela de Computación, Laboratorio MoST, Universidad Central de Venezuela, Caracas, Venezuela
[2] Escuela de Matemática, Laboratorio MoST, Universidad Central de Venezuela, Caracas, Venezuela

Abstract: Domain Engineering (DE) is crucial to determine limits and feasibility of a Software Product Line (SPL), or family of similar products or systems sharing common and reusable elements or core assets in a domain or market sector. The main goal of this work is to present an industrial experience centered on the PLScope phase of DE, considering a Quality-driven Domain Engineering Process (Q-DEP), using a bottom-up strategy based on the study of the enterprise existing products, to reduce the effort in the subsequent DE phases. This approach is more low cost and light weighted than the proactive (top-down) approach; however, both are recommended for SPL development by the ISO/IEC 26550 and Software Engineering Institute frameworks. Q-DEP has been applied successfully on a small-medium size Mobile Computing (MC) consortium that wanted to migrate to SPL. However, SPL development requires a huge engineering effort to build its main reusable artifact, the Reference Architecture (RA), or instantiable schema to derive concrete SPL products. RA is a highly abstract software architecture defined by components and connectors, holding placeholders to perform instantiation to derive new products. MC is facing increasing software demand, being a development based on information technology, fast multimedia transmission via computer or any wireless connected mobile device. MC does not deal with complex systems, but it must withstand very fast development/delivery issues and priority quality requirements such as efficiency, portability, usability and availability. Developers' teams work independently and loose effort in programming resources. The MC consortium wanted to build a first asset repository, to start migrating to SPL; construction of RA was not planned at the project start. However, the domain and existing products study was crucial to build the first asset repository, and our bottom-up process reduced the effort of the subsequent phases, hence RA was relatively easy to build.

Keywords: Domain Engineering; Software Product Line (SPL); Mobile Computing; Reference Architecture; Asset Repository; Q-DEP; bottom-up strategy

I. INTRODUCTION

Software Product Lines (SPL) for a particular domain or market sector, is an approach to industrial software development that provides massive personalization of individual solutions, sharing elements from a repository of reusable software assets, organized into a *Reference Architecture (RA)* with an instantiable schema, to derive concrete products of the SPL family [1][2][3][4][5]. A *domain* is defined in [6] as the minimal set of properties describing precisely a family of problems in which a computational application or system is involved for their solution. *SPL Engineering (SPLE)* is a huge and costly process since it involves the construction of an evolutionary RA. Two main lifecycles drive this "heavy" *SPLE* process*: Domain Engineering (DE*), where the *RA* and the *Asset Repository* are constructed, and *Application Engineering (AE*), where the RA schema or *variability model* is instantiated to derive new products [5] and the Asset Repository is also updated. In this work, DE and its first phase, the *Product Line Scoping (PLScop)* as recommended in the new standard ISO/IEC 26550 [5] will be considered. It takes into account the study of existing products built by the enterprise or available on the market, which is the base of our RA *extractive bottom-up* development strategy, derived from architectural-centric approaches. The Software Engineering Institute (SEI) [31] mentions that the reactive/extractive approach has the advantage of a much lower cost to start an SPL migration because the core asset base is not built up-front, and an existing product can be extended with new features to incrementally get the RA; it is much used in industrial practice and it contributes to reduce the huge effort required by the subsequent DE phases as it has been observed in [17]. The reactive approach considers one existing product that is extended to conform a SPL; the extractive approach instead considers a refactoring of several similar existing products to identify common and variant components to achieve the RA. The PLScop phase [5] deals with a broad capture of domain knowledge, to delimit the SPL application ambit, including risk analysis, economical feasibility and identification of the SPL family of products to be constructed. PLScop becomes necessary, since the knowledge on the SPL domain should be captured early, to

reduce the major effort concentrated on the subsequent phases of *Domain Requirements Engineering (DRE)* and *Domain Design (DD)*, where RA is actually built. Quality assurance should also be taken into account early, because quality related to domain *functional (FR)* and *non-functional requirements (NFR)* is responsible in a major degree of the SPL variability; however most of the approaches leave it to the DD phase. Our basic idea is to consider software components, often-common components that respond to precise quality goals, provided by others, often-variant components. Traceability of *functional (FR)* and *non- functional (NFR)* is achieved by construction, thus guaranteeing the RA evolution.

The main goal of this paper is two-folded; on one hand an industrial experience in the *Mobile Computing (MC)* domain, applying successfully a quality driven "semi-agile" light weighted DE process is related. On the other hand, the complete process that has been defined, applied and illustrated with the industrial case study is presented.

The fast world-wide network connection responsible of the so called "globalization", gave origin to MC software development [7][8], an approach highly based on *Information Technology (IT)*, allowing fast multimedia (data, voice, video) transmission via a computer or any wireless connected mobile device. Mobile software development involves widespread use of IT tools, often free and open-source support platforms, toolkits, etc., that can change rapidly over time. The new *Mobile Software Engineering (MSE)* discipline emerges [8], with the main trends of mobile communication, mobile software and hardware. In particular, mobile software is characterized by: - *Web Applications (Web Apps)* displayed on the user's device through a browser and executed on a remote server, - *Native Applications (Apps)* designed and developed for a particular device, but downloaded and installed by the user on his device, and – *Hybrid applications*, exposing Web Apps contents in Apps formats. Mobile software is not intensive software and does not involve thousands of lines of code, but it must be developed very fast to satisfy a huge market demand and its dynamic nature. Moreover, it must have a certain quality level, expressed by quality requirements such as usability, efficiency and availability, being this crucial for the immediate acceptance or rejection of the product; the wide spread use of rapidly implemented and evolutionary IT characterizes this kind of development, being client satisfaction one of the main goals. However, the success of a software application, mobile or not, depends not only on its functional suitability representing adequate FR, but also on the satisfaction of the quality properties representing NFR, required by these functionalities to have a suitable behavior [8][9][10].

Good practices of software engineering, as recommended in [11] are not widespread in MSE [8]. Important aspects to be considered relative to the quality properties already mentioned, are: - the integration of hardware and software, - limited resources such as data storage and display for the *User Interface (UI)* on reduced size screens with different resolutions and devices, - frequent changes in IT, - use of interoperability standards, and traditional quality requirements, such as – portability to different platforms, - reliability w.r.t. availability of Internet connection, - security to guarantee access control, and finally - efficiency in the UI display and process services to execute functionality. In view of the growing demand of mobile applications from 2007[1] with the iPhone success, a survey in [8] on main practices used in mobile software development, reported the following important aspects:

1. Applications are "small", few thousands of lines of code in average, involving only one or two developers.
2. There is quite a difference between Apps and Web Apps.
3. No development methods are used in this domain.
4. Developments are not documented and very few metrics are registered.

Notice that points 3 and 4 are against software engineering best practices[2]. Nevertheless, lots of commercial platforms are available to develop mobile applications, and as it will be shown later on, they allow guaranteeing to a certain extent the accomplishment of priority quality properties required by the functionalities. With respect to point 3, the so called "agile" methods, practices and techniques [12][13][14], usually driven towards a rapid UI development to achieve client satisfaction, are recommended for MSE [8][15].

We faced the problem of a small-medium sized (more than 20 employees) mobile software enterprise that wanted to identify reusable software assets to build a first repository of reusable software components (modules, toolkits, APIs, mechanisms, etc.). They were working on three different sub-domains of MC (financial transactions, healthcare systems and entertainment contents), and the programming effort was triplicated, since it was not known which semantically similar components were actually used in which sub-domain. Our proposed solution was a "slow" migration towards a MC product line, starting with the constructions of a high-level Core Asset Repository, holding reusable and variant components used within the different products developed by each sub-domain of the enterprise. The project at first did not pretend to construct a RA for the enterprise SPL and only the PLScop phase was to de developed. However, in order to construct a first draft of the repository, a complete DE process had to be done, but they had no time and no human resources, dedicated mostly to development, to employ into such a huge task. In consequence, a "light" or "semi-agile" DE phase of SPLE was proposed, including *PLScop*, *Domain Requirements Engineering (DRE)* and *Domain Design (DD)* phases [5]. However, for the product line migration to be successful, the architecture and other core assets must be robust, extensible, and appropriate to future product line needs [31], and a RA could be constructed in this way for each sub-domain, profiting from the extractive bottom-up design. The process should start by reengineering the available information to construct the architecture of each product/system within the sub-domain, studying similarities and differences among components on the basis of interviews and meetings (architecture documentation was absent, as usual in industrial practice). Based on the extractive strategy, the bottom-up approach was applied, meaning that more than one product built by the enterprise had been considered, to construct automatically an initial *Candidate Architecture (CA)* represented by a connected graph [10], by the graph union of the existing products' architectures,

[1] https://wikipedia.org/wiki/History_of_iPhone
[2] http://technav.ieee.org/tag/4655/best-practices

designed at a high abstraction level, also represented by graphs. To achieve this, the process *Q-DEP: Quality-oriented Domain Engineering Process* was defined, adapted from [24], and applied during a period of six months.

Q-DEP concerns the main phases of the DE lifecycle including a first assessment for an Asset Repository; it does not involve source code reengineering; the information on the products developed by the enterprise was captured via an "agile" practice, from a quiz and several interviews with the developer teams' leaders, until a consensus on the configuration of the existing products' architectures, in terms of high-level description of components and connectors [16], is achieved. Notice that similarities among the products developed are assumed to exist because they belong to the same domain/sub-domain. We recall that in general industrial practice the logic view of the software architecture is not developed, and often the only documentation available are deployment diagrams representing physical nodes where components are running [32]; however, it is required to construct the RA in the SPLE approach, to identify *common components* and *variant components* (they do not appear in all products) involved. Fine grained goals of this work are: a) design an RA for each sub-domain, according to a bottom-up approach based on the refactoring of existing products' available documentation, built and used by the enterprise, b) conform a first draft of the assets, a list of common and variant components that are used within the enterprise products, and c) Illustrate the stepwise application of the process considering only one of the sub-domain for this presentation.

In addition to this introduction, this work is structured as follows: Section II discusses some related works; Section III describes the Q-DEP process (phases, activities); Section IV shows Q-DEP applied to the Healthcare Information Systems (HIS) case study of the sub-domain of the *XX-MC* enterprise. Finally the conclusion and perspectives are presented.

II. Related Works

Very few scientific research works were found discussing MC and even less about SPL for mobile computing. Few use a bottom-up strategy, which we consider a much more pragmatic, fast and practical approach to SPL development, since the fact of having one or more existing products available on the market or developed within the enterprise, is a more common situation than to build the SPL from scratch (top-down approach) [10][31]. The new standard SPL Reference Model [5], even favoring a global proactive top-down approach, introduces explicitly an initial scoping phase to handle this problem, including a product portfolio that can be built from products on the marketplace and/or from the enterprise own products, using bottom-up techniques. In this work, we have adopted this strategy, also to reduce the effort in the subsequent DE phases, including however quality issues, even in this first phase, differing from [5], which delegates these activities to the late DD phase [17]. The following works will be discussed:

- A method is proposed in [18] for UI development in the MC domain, called GeMMINi; *Model Driven Design (MDD)* is used to model transformations, combined with feature modeling [19]. The method is very informally specified; it describes at a high abstraction level UI requirements and the device variants, applying model transformations and generating code to obtain Apps for UI on different devices. A catalogue of patterns is defined to translate UI abstract concepts into concrete device specifications; it is used to configure the transformations; the user interaction is specified into "units" with UML class diagrams, containing the data structure descriptions. The specification of properties and variants of the devices are provided by an ontology-based feature model [19][20], to specify some aspects of the interaction. As it is usual in feature models approaches, no RA is considered.

  This paper involves the DE and IA cycles of SPLE; however, just the UI component is treated, and recent mobile computing development toolkits already exist to conform the UI for each device, such as IONIC[3]. Only functional variability is handled, because the usual feature model only considers this aspect, even if lots of works have been done to include also non- functional variability [10]. Our approach with Q-DEP is more fine-grained; we handle the refactoring of all main components obtained from the architectural configurations of similar products within the enterprise. We don't use feature modeling nor MDD, using a scenario-based approach instead [21]; an initial CA is automatically constructed, by the "union" of the graphs representing the architectures of the existing products, establishing a first variability model that can be completed with additional information. We retrieve traceability among FR and NFR on the bases of tables representing scenarios [21], where each component or service solving a quality property required by a functionality, is specified.

Previous works [10][22][23][24] have evolved to define the present Q-DEP process. They will be discussed in what follows:

- The work in [22] treats the robotics domain. Reengineering techniques are mostly used, from code or documentation, to reconstruct the architecture of existing products in the enterprise. RA is built manually and directly, without considering an intermediate architecture. The justification of the satisfaction of NFR is very informal and standards are not used to specify quality properties. It has inspired our present research.
- The idea of representing a software architecture [16] by a connected graph was developed in [23], to perform the automatic "union" of the refactored architectures of similar existing products (belonging to the same domain), according to a bottom-up strategy inspired in [22]; in this way a first CA was constructed, showing common and variant components obtained from the products considered. Quality properties are grouped as scenarios in the Extended Quality Model (EQM) Table, and were specified by the standard ISO/IEC 25010 quality model [9]; EQM contains components and their required quality properties; however traceability among FR and NFR was not completely justified and the variability model was established according to an optimization process. The process was applied to the robotics domain, as in [22].

[3] https://ionicframework.com/docs

- The process originally defined in [21] was reformulated in [10] into a new process NF-VAR, to construct RA, combining the extractive bottom-up strategy with goal-oriented techniques [25], using the *Softgoals Interdependence Graph (SIG)* diagram, to complete the CA obtained automatically [23], with new components introduced to satisfy NFR. NF-VAR was applied to the Healthcare Information Systems (HIS) domain, using three open-source market products, OpenEMR[4], PatientOS[5] and Care 2X[6]. However, the use of the SIG was not straightforward, even if tools such as GRL[7] were available; it is difficult to handle complexity with this diagram, which does not offer a standard notation.
- The *QuaDRA (Quality-oriented Design of Reference Architecture)* process was specified for SPL in [24]; the new ISO/IEC 26550 standard defining the SPLE Reference Model [5] was followed to include PLScop, the first DE phase, to reduce the effort in the subsequent DRE and DD phases of DE. The NF-VAR process defined in [10] fitted well into PLScop, to construct the SPL product portfolio and to produce automatically CA and the EQM Table; the Domain Scoping phase taken from [26], was used to complete CA with additional information captured by different stakeholders viewpoints, including quality as a new intrinsic facet to describe stakeholders viewpoints. In this way, business processes specified in BPMN were integrated to the process, attaching quality requirements to each functional task. In this way the use of the SIG was avoided.

  Q-DEP was adapted from [24]: business processes specification were not included to have a more "light weighted" DE process. Requirements elicitation was done using "agile" practices by interviewing stakeholders and performing meetings to achieve a consensus on components and connectors conforming the architectural configuration [16]. Our process is highly based on the provide/require scenarios tables for quality properties traceability, maintaining the CA automatic construction by a bottom-up strategy.

III. Q-DEP: Quality-driven Domain Engineering Process

A PL Scoping phase including Domain Analysis activity and some technics inspired from agile methodologies are focused in Q-DEP to achieve a "light" DE process; it does not consider business processes as in [24], being more oriented towards a "naïve" but practical approach of DE, involving direct interaction with project leaders. Three main phases of the DE lifecycle have been considered: *1. PLScop (SCOP)* with main activities: – Build Product Portfolio (ProdPort), - Agile Domain Analysis (ADA), - Assets Identification (AssetT), and Build Global Assets (GLAsset); *2. Domain Requirements Engineering (DRE)* with main activities: – Build Candidate Architecture (CA), Build the UML representation of CA, and Build EQM, and - *3. Domain Design (DD)* with main activities: – Build Reference Architecture (RA), – Elaborate General Assessment Document (GAD). The complete process is specified in Figures 1 and 2; the main effort is concentrated in Phase 1 SCOP, but this will reduce the work in the subsequent phases. When ADA is mentioned [27], we do not pretend to develop or use a complete "agile" methodology [12] [13][14], but just use some basic technics such as interviews and meetings with main project leaders. We want to achieve a consensus on the enterprise product components, connectors and architectural solutions. The number of meetings will depend on the complexity observed during the first meeting to elicitate the information on the enterprise domain/sub-domain(s). We recall that even if agile methods are suggested for the MC domain, we face an SPL context where a family of similar systems is built, and agile methods seems to be better suited for single systems' development [28].

To achieve a *variability model* imbedded into the RA [5] showing the placeholders to be intatiated, the products' *common components* are identified from a semantic viewpoint, considering the accomplishment of functionally similar tasks; the *variants components* will then be glued also according to the similarity of their tasks, into categories called *variation points* [3], which are sets of variant components.

IV. Application of Q-DEP to the Healthcare Information Systems Sub-domain

Q-DEP was applied successively to three sub-domains (financial transactions, healthcare information systems and entertainment contents) of the XX-MC enterprise. Recall that the enterprise problem was to determine the main components, modules, support platforms and/or toolkits that had been used in product developments, to avoid repeated programming efforts. The application of the process was done for the three sub-domains during 6 months; about a hundred components were identified as assets for the three sub-domains.

Q-DEP will be applied here to the Healthcare Information Systems (HIS) sub-domain.

The general HIS architecture is a hybrid event-based style, SOA[8]/Layers, following a client-server model for distribution and communication. HIS must facilitate transparent sharing of different kinds of medical information such as EHR and laboratory and imaging results, offering also telemedicine services that can be performed on-line at remote locations, with wide support of information technology. The use of standards such as HL7[9], HL7 CDA, LOINC[10], and DICOM[11] are mandatory for interoperability of HER, and laboratory and imaging results. Nevertheless, in actual medical practice, SPL for HIS have not yet been completely defined, developed and adopted; the lack of agreement on medical standards and psychosocial issues makes difficult the interoperability of EHR, and HIS general adoption is still difficult, even if specific laws and regulations towards these goals have been promulgated worldwide.

[4] https://www.open-emr.org
[5] https://sourceforge.net/projects/patientos
[6] https://www.care2x.org
[7] Goal-oriented Requirement Language, https://www.cs.toronto.edu/km/GRL
[8] Service-Oriented Architecture
[9] https://www.hl7.org
[10] Logical Observation Identifiers Names and Codes
[11] Digital Communication in Medicine

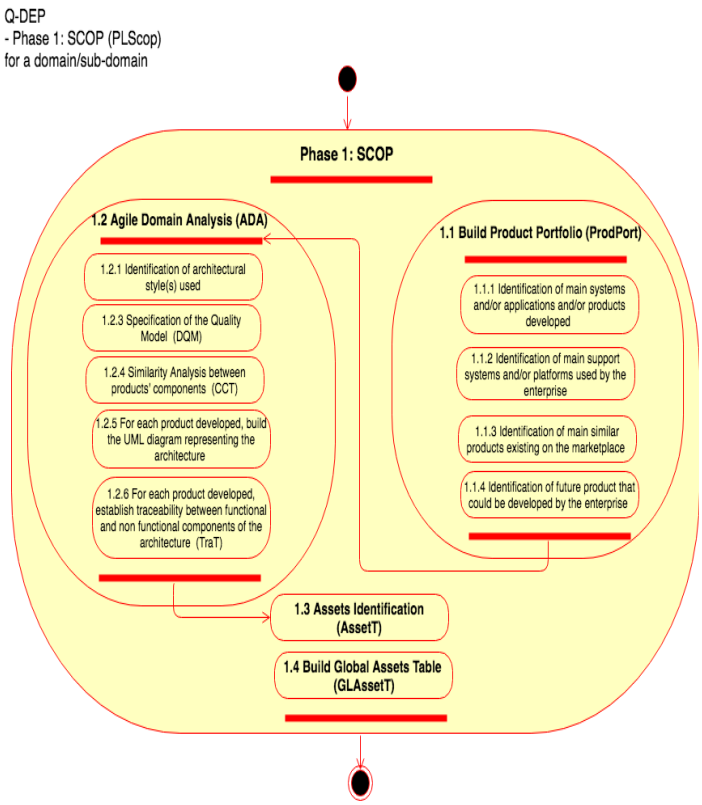

Figure 1: Q-DEP – Phase 1 SCOP – UML[12] [29] Activity Diagram

We will be limited here to basic HIS functionalities, handled by most open-source HIS, and with the physical doctor-patient encounter episode.

A. Phase 1. SCOP Applied to the HIS Sub-domain

SCOP input:

- *One quiz and two meetings:* the following general information was elicitated:
- Business goals:
- Facilitate information management to doctors and provide aggregate value to patients with on-line IT via Web, on stand-alone or mobile devices.
- General requirements for Patients:
  - Personal data can be modified by authorized doctors or by patient.
  - A Patient can be registered into the system by a doctor: he cannot enter the system without receiving and answering the invitation generated by the doctor.
  - Images are registered by image laboratories, or other image centers (to be developed): - Data persistency: a package is offered for the period that a patient data has to be kept; - patient is notified for payment.
  - Doctor associates data to patient during the first appointment, allowing him to consult/modify his personal data.
- General requirements for Doctors:
  - He can register to the system personally or have been invited by a patient when he associates him to his data. If he will not answer, he will not be registered by a patient invitation.
  - If no Internet connection is available, manual data transcription (24x7x365) should be managed
- *Business Rules (BR):* use of the proprietary AWS (Amazon Web Services) provider to have *Infrastructure as a Service (IaaS)* cloud configuration, use of *PostgreSQL Database Management System (DBMS)*, use of *Java*[13] language & related platforms

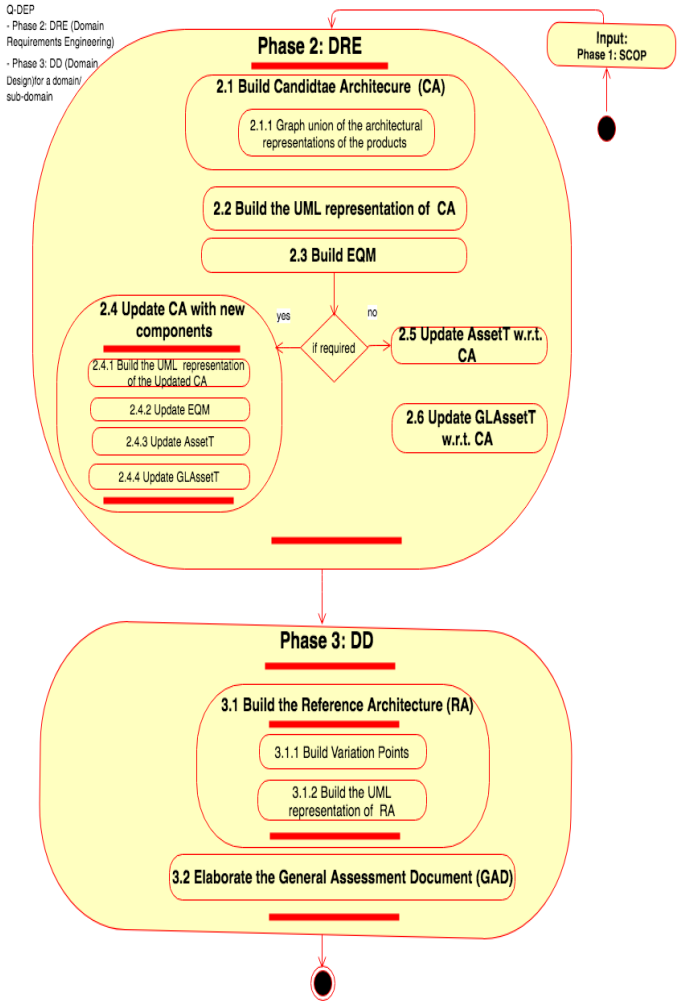

Figure 2: Q-DEP – Phase 2 DRE & Phase 3 DD – UML Activity Diagram

1. Build Product Portfolio (ProdPort) - Activity 1.1:

- Products identified were: *Web Application (WebApp) – for* patient attention services, general medical practice, administration; displayed via a browser and executed through AWS; *Native Application (NatApp)* free download from Apple Store (iOS) and/or Paly Store (Android) and displayed on the client device, offering the same services as WebApp, also through AWS. The UI of

[12] Unified Modeling Language

[13] Sun Microsystems

both products interacts with the AWS cloud services for data process, retrieval and storage; two main components are distinguished: *FrontEnd (Presentation Layer)* and *BackEnd (Process and Data Layers)*; Table I shows main NFR and main support IT mechanisms used, for each product in each layer; a similar product available on the market was *Viewmed online*[14].

- Future products or extensions that could be developed: - assure EHR interoperability with standard formats, like HL7; - Graphical imaging management; - assure EHR availability, since it depends on the cloud; - on-line help to diagnosis; - use of digital electronic payment media.
- Products Portfolio (ProdPort): WebApp, NatApp.

Table I: NFR - FrontEnd – Presentation Layer

FrontEnd NFR	WebApp – IT tools satisfying NFR	NatApp – IT tools satisfying NFR
usability	CSS3[15] for page styles; it depends on the page design	Java and Swift[16] languages for secure messages services
portability	AngularJS[17]-JavaScript[18], HTML5[19]	Swift for iOS and Java for Android SDK [20]
maintainability (modifiability)	AngularJS, with MVC[21] to separate Presentation and Process layers	BackEnd - Ruby-on-Rails[22] provides the RESTful Web service with MVC to send data to NatApp, to separate Presentation and Process Layers
Resources utilization (time efficiency)	Bootstrap[23] FrontEnd framework	Java and Swift languages
Security (authenticity, confidentiality, integrity)	Module developed by the enterprise for access control and role policy	Module developed by the enterprise for access control and role policy

Table II: NFR - BackEnd – Process Layer

BackEnd NFR	WebApp – IT mechanisms satisfying NFR	NatApp – IT mechanisms satisfying NFR
security (authenticity, confidentiality, integrity)	HTTP/HTTPS[24]; system access control, roles policies to access EHR are in a separate module developed by the enterprise	same
portability	Ruby-on-Rails: open source platform and Ruby language; it can work without a specific dada base	same
reliability (robustness)	Ruby-on-Rails	same
reliability (availability-persistency)	PostgreSQL DBMS for data persistency via AWS; the system availability depends on the AWS connection	same
modifiability	Ruby-on-Rails	same
interoperability	AWS cloud services	same

[14] https://www.viewmedonline.com
[15] https://developer.mozilla.org/es/docs/Web/CSS/CSS3
[16] https://www.swift.com
[17] https://angularjs.org
[18] https://www.javascript.com
[19] http://www.w3.org/TR/2014/REC-html5-20141028
[20] https://stuff.mit.edu/afs/sipb/project/android/docs/sdk/index.html
[21] Model, View Controller (GOF pattern), https://wikipedia.org/wiki/Model–view–controller
[22] https://rubyonrails.org
[23] https://getbootstrap.com
[24] Internet Communication Protocols

2. Agile Domain Analysis (ADA) – Activity 1.2:

- *The architectural style(s) used were:* event-based/layers, with a client-server model for distribution and communication.
- *Priority of FR and NFR:* - Priority NFR: security (1), interoperability (1), availability (1), efficiency (time and resources) (2), usability (for the FrontEnd design) (2), maintainability (modifiability) (3), correctness-precision (3), where $1 \leqslant \text{priority} \leqslant 3$; - Priority FR: Appointments and turns control, HER Management (1), Support for Doctoral Practice, for access roles management.
- *Quality Model (DQM) by [ISO 11], considering FR&NFR (Tables I and II) BR&IT, quality properties from architectural styles:* security, portability, maintainability (modifiability, reuse), usability, efficiency (time, resources, scalability), reliability (availability, maturity-robustness), functional suitability (correctness-precision).
- *Similarity Analysis between products' components:* It is shown in Table III, for each product; "R" represent the relation or connector between two components; common components are outlined in grey; the name of the components are taken from Tables I and II; components are labeled sequentially according to the layer and the sub-domain:

Table III: CCT Table showing Components and Connectors

FrontEnd - Presentation Layer		**BackEnd - Process and Data Layers**
WebApp	**NatApp**	**WebApp & NatApp**
a1. FrontEnd	a1	**b1. AWS – BackEnd**
a2. WebPortal	-	**b34. Ruby-on-Rails**
-	a13. UI	**b35. Patient**
a4. In Browser	-	b36. Register Patient
a5.In AngularJS	-	b37. Appointments and turns
a6. In HTML5	-	**b38. EHR Management**
a7. In CSS3	-	**b39. Medical Practice**
a8. In Bootstrap	-	b40. Register Doctor
-	- a14. Java-Android	b41. Examinations
-	- a15. Swift-iOS	b42. Recipes and Medicines
		b43. Medical Reports
		b44. Administration
		b45. Billing
		b46. Laboratory
		b47. Insurance
		b48. Security
		b49. Lab Info
		c1. DBMS
		c2. PostgreSQL
Connectors		**Connectors**
a1Ra2	-	b1Rb34
-	a1Ra13	b34Rb35, b35Rb38, b34Rb39,
-	a13Ra14	b34Rb44, b34Rb48, b34Rc1
-	a13Ra15	b35Rb36, b35Rb37, b35Rb38,
a2Ra4	-	b35Rb39, b35Rb44, b35Rb48

a2Ra5	-	b38Rb39, b38Rb48
a2Ra6	-	b39Rb40, b39Rb41, b39Rb42,
a2Ra7	-	b39Rb43, b39Rb44, b39Rb48,
a2Ra8	-	b39Rc1,
a2Rb1	-	b44Rb45, b44Rb46, b44Rb47,
-	a13Rb34	b44Rb48, b46Rb49
		c1Rc2

- *Architecture of the main products developed:* it is built using the identified components and connectors for each product from CCT Table (see Table III, Figures 3 and 4).
- Notice that in the FrontEnd, excepting the main component *a1. FrontEnd*, there are no common components; user-interfaces are different because one is a Web page for WebApp and the other is a classic stand-alone UI for NatApp; all components in the BackEnd are common.
- For each product developed by the enterprise, the traceability between the components of the architecture is established in the TraT Table (see Table IV).
- Notice also that a separate table should have been built for each product, but in this case only the FrontEnd was different and both tables were integrated to abridge the presentation.

3. Assets Identification - Activity 1.3:

   Main common and variant assets for the sub-domain are determined building the AssetT, see Table V;

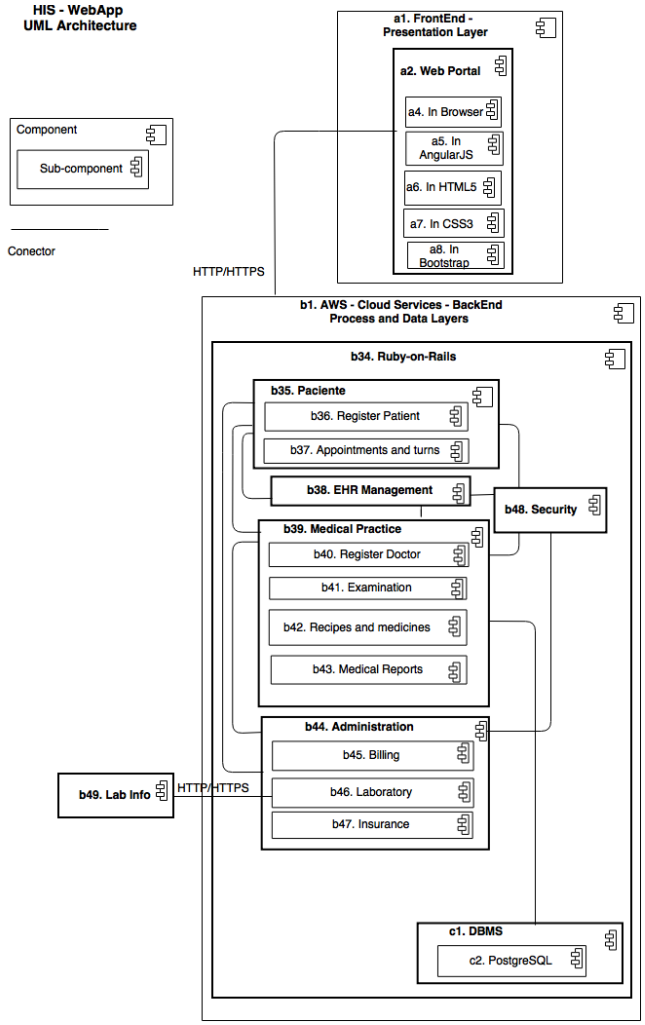

Figure 3: Architecture of WebApp Product Expressed in UML 2.0

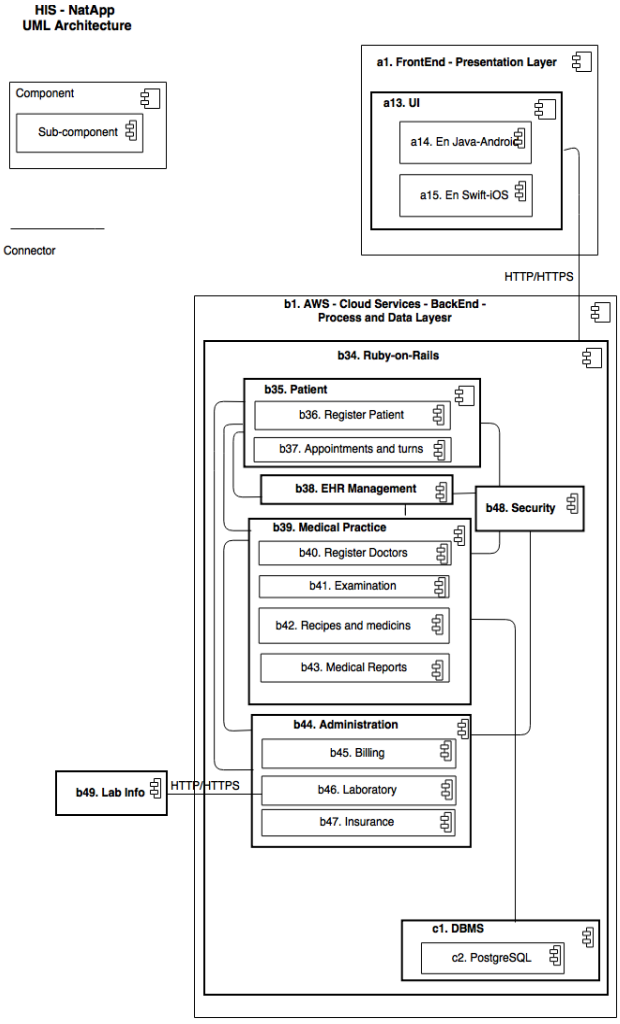

Figure 4: Architecture of NatApp Product Expressed in UML 2.0

Table IV: TraT Table showing Required Quality Properties and IT Components Providing them

WebApp and NatApp Components	Required Quality Property	Provided IT Tool to satisfy Quality Property	Description/Comments
a1. FrontEnd	- availability	- a4. In Browser	- depending on the browser availability
- a2, Web Portal	- usability	- a6. In CSS3	- cascade style for HTML documents
	- modifiability	- a5. In AngularJS	- open framework, JavaScript-MVC
	- efficiency	- a7. In Bootstrap	- open framework for development
	- portability	- a8. In HTML5	- markup language version 5
- a13. UI	- portability	- a14. In Java-Android	- display on different OS with Java and Swift
	- usability	- a15. In Swift-iOS ---	- UI is developed in Java and Swift languages; usability depends on the design, it will not be assured by the architecture
	- modifiability	- b34. Ruby-on-Rails	- Web service *RESTful* of Ruby-on-Rails is provided by the BackEnd
	- availability	- 100% on device	- it is a stand-alone UI; downloaded form App Store (iOS) or Paly Store (Android); the execution of functionalities depend on the availability of the cloud services
	- efficiency	- b34. Ruby-on-Rails	- open-source developing platform
	- security	- b34. Ruby-on-Rails	- module developed by the enterprise for access control and role policy, in the BackEnd
b1. AWS - BackEnd	- portability - interoperability - modifiability	- Cloud services provider	- IaaS; AWS is compliant with the quality of cloud services
	- security	- HTTP/HTTPS	- network communication protocols
	- reliability (availability-persistency, robustness)		
- b34. Ruby-on-Rails	- reliability (availability-persistency, robustness) - portability-modifiability	- Development Platform	- open-source development platform, includes components and sub-components
- b35. Patient	- security (authenticity, confidentiality, integrity)	- b48. Security	- Module developed by the enterprise for access control and role policies
	- availability-persistency	- c2. PostgreSQL	- Microsoft relational DBMS; on-line availability depends on Internet connection since it runs on the cloud; b16. Ruby-on-Rails holds mechanisms for portability to Java objects
- b36. Register Patient	- same as b34	- b34	
- b37. Appointments and turns	- same as b34 + correction-precision	- b34	- handled by b34. It was not specified as a specific component or module
- b38.HER Management	- security (authenticity, confidentiality, integrity)	- b48. Security	- Module developed by the enterprise for access control and role policies
	- availability-persistency	- PostgreSQL	- Microsoft relational DBMS; on-line availability depends on Internet connection since it runs on the cloud; b16. Ruby-on-Rails holds mechanisms for portability to Java objects
	- efficiency (resources-scalability)		
	- interoperability	--- absent --	- to be developed – integration of IT tools for HL7
- b39. Medical Practice	- availability-persistency	- same as b38	
	- security (authenticity, confidentiality, integrity)	- b48. Security	- Module developed in Java by the enterprise for access control and role policies
- b40. Register Doctor	- same as b39	- same as b39	- b40, b41, b42, b43 are sub-components of b39
- b41. Examination	- same as b39	- same as b39	
- b42. Recipes and medicines	- same as b39	- same as b39	
- b43. Medical Reports	- same as b39	- same as b39	

- b44. Administration	- security (authenticity, confidentiality, integrity)	- b48. Security	- Module developed in Java by the enterprise for access control and role policies - handled by b34. It was not specified as a specific component or module
	- correctness-precision	- b34	- handled by b34. It was not specified as a specific component or module
	- efficiency (response time)	- b34	- handled by b34. It was not specified as a specific component or module
- b45. Billing	- interoperability	- Adobe Acrobat	- pdf files
- b46. Laboratory	- the same as b44	- the same as b44	- b45, b46, b47 are sub-components of b44
- b47. Insurance	- the same as b44	- the same as b44	
	- the same as b44	- the same as b44	
- b48. Security	- security (authenticity, confidentiality, integrity)	- Module	- developed in Java by the enterprise for access control and role politicies
- b49. Info Lab	- availability-persistency	- external module to AWS	- it contains laboratory information; used by b46
c1. DBMS	- all DBMS quality properties hold	- Database Management System	- Microsoft relational DBMS; b16. Ruby-on-Rails holds mechanism for portability to Java objects; quality properties are provided by DB mechanisms.
- c2. PostgreSQL	- the same as c1	- same as c1	

Table V: AssetT Table showing main Assets used by Products WebApp and NatApp

Layer	WebApp and NatApp Components	Comments
FrontEnd - Presentation Layer	- a1. FrontEnd - a2. Web Portal - a4. In Browser - a6. In CSS3 - a5. In AngularJS - a7. In Bootstrap - a8. In HTML5 - a13. UI - a14. In Java-Android - a15. In Swift-iOS	- No common assets
BackEnd- Process Layer	- b1. AWS – cloud services - b34. Ruby-on-Rails - b35. Patient - b36. Register Patient - b37. Appointments and turns - b38. HER Management - b39. Medical Practice - b40. Register Doctor - b41. Examination - b42. Recipes and medicines - b43. Medical Reports - b44. Administration - b45. Billing - b46. Laboratory - b47. Insurance - b48. Security - b49. Info Lab - b80. InteropEngine - b81. Imaging	- All assets are common to WebApp and NatApp - b35, …, b48 are common components developed by the enterprise - New component - New component - Internet communication protocols, variants for WebApp and NatApp
Communication Protocols	- HTTP/HTTPS	- Common for layers' interface
BackEnd - Data Layer	- c1. DBMS - c2. PostgreSQL	- Common to WebApp and NatApp; portability to Java objects is responsibility of b16. Ruby-on-Rails

SCOP output:

ProdPort (WebApp, NatAPP), FR (Table I), *NFR* (Table I), *CCT Table* (Table III), *TraT Table* (Table IV), *AssetT Table* (Table V), *WebApp architecture* (Figure 3), *NatApp architecture* (Figure 4).

B. Phase 2. Domain Requirements Engineering (DRE) Applied to the HIS Sub-domain

1. Build Candidate Architecture (CA) – Activity 2.1: Automatic construction of CA by the union of the graphs representing the products' architectures, using

the information provided by SCOP artifacts: ProdPort, CCT and TraT tables.

2. Build UML representation of CA –Activity 2.2:
It is shown in Figure 5. variants are a2 and a13 on the FrontEnd and the connectors to the BackEnd; all other components are common.

3. Build table EQM – Activity 2.3:
Integrate the TraT tables for each product with provided/required quality properties and possible constraints, see Table VI. EQM shows a different CA view, documenting possible scenarios of IT components satisfying quality requirements. In this case the BackEnd is similar for both products and the EQM table is very similar to the TraT table; only the FrontEnd will be shown in Figure 6, as an example of the EQM Table.

4. Update CA with new components – Activity 2.4:
it was not necessary in this case because there were no new components.

5. Update AssetT – Activity 2.5:
with respect to CA which has been automatically built; the new components for interoperability and imaging will be added, they are placed directly in Table V (AssetT Table) to abridge the presentation.

6. Update GLAssetT Table – Activity 2.6:
GLAsset = AssetT in this case, since only one sub-domain is considered.

Table VI: EQM Table

CA Components	Quality Property required by Component	IT Components satisfying Quality Properties	Description/Comments/Constraints
a1. FrontEnd			**Process and Data Layers**
- a2. Web Portal (variant)	- availability	- a4. In Browser	- depending on the browser availability
	- usability	- a6. In CSS3	- cascade style for HTML documents
	- modifiability	- a5. In AngularJS	- open framework, JavaScript-MVC
	- efficiency	- a7. In Bootstrap	- open framework for development
	- portability	- a8. In HTML5	- markup language version 5
	- security	- b34. Ruby-on-Rails	- accessed at logging the system
		- a14. In Java-Android	- display on different OS with Java or Swift
- a13. UI (variant)	- portability	- a15. In Swift-iOS	depending on the device OS
		---	- UI is developed in Java and Swift languages; usability depends on the design, it will not be assured by the architecture
	- usability		
		- b34. Ruby-on-Rails	- Web service *RESTful* (MVC) of Ruby-on-Rails is provided by the BackEnd
	- modifiability		
		- 100% on device	- it is a stand-alone UI; downloaded form App Store (iOS) or Paly Store (Android);
	- availability		- the execution of functionalities depend on the availability of the cloud services
		- b34. Ruby-on-Rails	- open-source developing platform
	- efficiency		
			- a2 and UI, including their sub-components, cannot be present together in a product configuration
	- security	- b34. Ruby-on-Rails	- Accessed at login

C. Phase 3. Domain Design (DD)

1. Build the Reference Architecture (RA) – Activity 3.1:

- *Build Variation Points* by grouping variant components performing similar tasks: there are three variation points for the FrontEnd: - <<*a16. Portal*>>, which includes different portals that can be built considering also the sub-variation point <<*a40. PortDevPlatforms*>> where choices of platforms could be made according to IT changes; - <<*a17. UInterface*>> that can include different stand-alone UI designs and also different IT choices; - <<*a18. UInterfaceConnector*>>, because *a2* and *a13* connect differently *to b1. BackEnd*: *a2* using *b4. AngularJs* who provides the separation of Presentation and Process Layers to get modifiability; *a13* uses instead *b34. Ruby-on-Rails* with the RESTful service for MVC to get also modifiability. It is clear that new IT mechanisms will continue to appear on the market, similar to those used by the enterprise and other cloud services providers besides AWS, and other developing platforms besides *b34. Ruby-on-Rails* could be adopted, hence two new variability points are included in the BackEnd, <<*b82. Cloud Services*>> and <<*b83. BEDevPlatforms*>>, because can be also changed, providing more genericity to this HIS RA.
- *Build the RA UML diagram representing the architecture*: it can be seen in Figure 6. New components that have been stated in Future products are included, *b80. InteropEngine* for HL7 standard and *b81. Imaging* to handle graphic imaging.

2. Elaborate the GAD Document – Activity 3.2:

The GAD document contains basically a summary of the artifacts obtained, limitations and recommendations. Among the limitation:

- The architecture documentation (components and connectors) [16] was almost absent for the enterprise, being reduced to just an incomplete deployment diagram; however it was found that now with the UML 2.0 architectural diagrams provided, deployment diagrams could more be easily designed, knowing explicitly all the components that were using.
- No standards were used.

Among the recommendations:

- Maintain updated all the tables and diagrams in case of changes, since they are the reusable assets, including the RA diagrams or documentation.
- Incorporate as soon as possible a standard format for EHR.
- The GLAssetT Table should be implemented as a "real" Core Asset Repository of the artifacts produced.

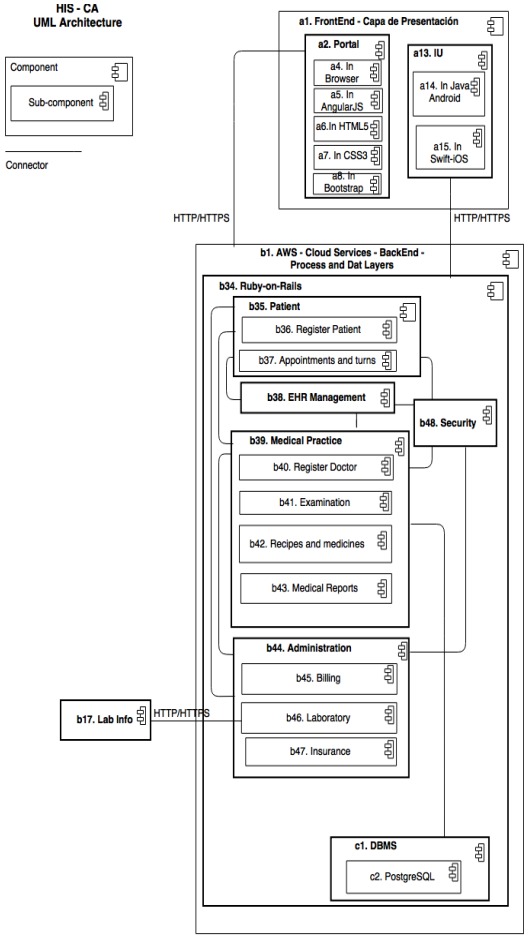

Figure 5: HIS CA Expressed in UML 2.0: Union of WebApp and NatApp Graphs' Architectural Representations

V. CONCLUSION

The Q-DEP extractive bottom-up quality-driven domain engineering process for SPL has been presented. Our aim was to relate the experience in applying a "light weighted" SPL engineering approach that can be effectively used in industrial practice. Q-DEP has been entirely developed for this experience, adapting the QuaDRA process [24] to a real industrial context. Both processes are centered on the first PLScope phase of DE to reduce the efforts in the subsequent phases, but Q-DEP does not use business processes specifications to identify main functionalities and constraints; using a "semi-agile" practice more adapted to the MC domain, it profits from interviewing stakeholders and performing meetings to arrive to an agreement on the existing products architecture. It was applied to the MC domain, facing an increasing market demand and where no software engineering good practices are yet employed, due to the rapid time-to market delivery requirement. The experience with a small-medium sized software enterprise was satisfying, three different sub-domains of MC were studied successively (financial transactions, healthcare information systems and entertainment contents), and the components of the products

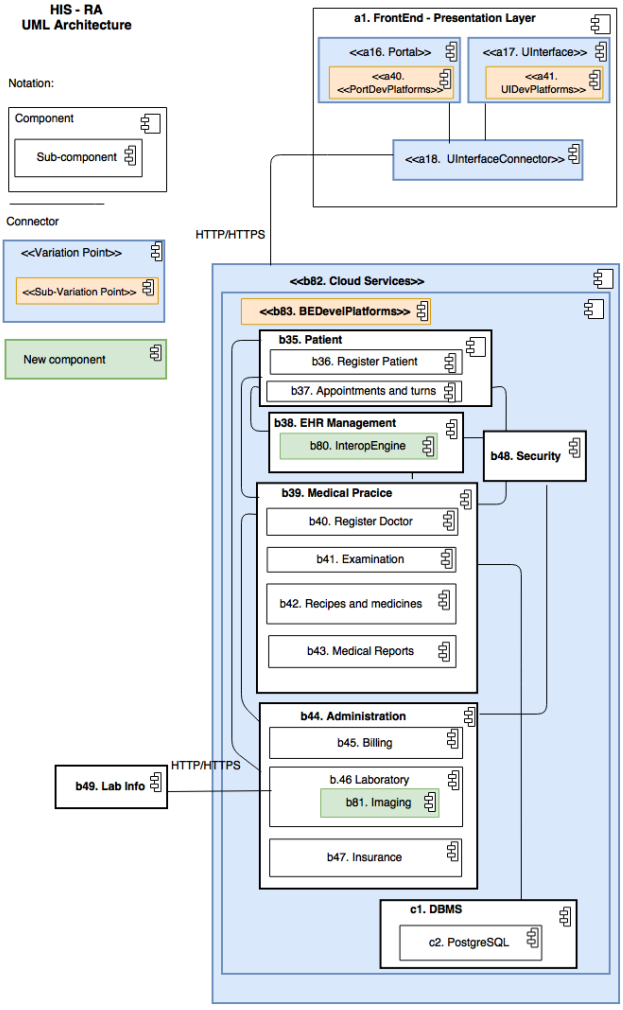

Figure 6: HIS-RA Expressed in UML 2.0

developed were identified by direct interaction with project leaders. We did not pretend to construct a complete SPL, but just to start a migration by building a first glance of the enterprise Core Asset Repository and the RA of the sub-

domains considered. In this paper we applied Q-DEP to the Healthcare Information Systems sub-domain, showing and explaining the construction of all the artifacts produced. In Q-DEP quality is considered from the start. Software development in MC is based on reusing lots of IT support tools, and this fact has been useful showing implicit satisfaction of the quality goals required by MC applications; developers use IT tools transparently, generally unaware of the quality aspects involved, but in this way the quality of their applications is assured "by construction" to a certain extent. As IT evolves quickly, there is hope that support tools will also improve and implicitly continue to guarantee the quality of mobile software. Another aspect that has to be signaled is that several design choices, such as the selection of the variation points, are mostly based on the architect expertise and on his vision of the domain evolution, but this is difficult avoid in architectural design. On the other hand, a limitation of Q-DEP is that automatic support tools are under construction and they are necessary to face complexity of intensive systems; besides, the integration of the different RAs of the three sub-domains into a global RA for the enterprise is still an ongoing work, a new sub-process has to be defined and integrated to Q-DEP.

Among the perspectives, we are working on an ontological approach to represent the RA [30] and the Asset Repository, to derive consistency rules for the AE cycle. A comparison of the ISO/IEC 26550 versions of 2013 and 2015 is also planned.

ACKNOWLEDGMENT

We wish to thank María Dolores Nardi, Aquilino Pinto, Carlos Alfonzo and Mary Gutiérrez, to make this work possible with fruitful comments and observations.

REFERENCES

[1] J. Bosch, *Design and Use of Software Architectures – Adopting and Evolving a Product-line Approach*, Addison-Wesley, 2000.

[2] P. Clements and L. Northrop, *SPL: Practices and Patterns*, 3rd edition. Readings, MA, Addison Wesley, 2001.

[3] K. Pohl, G. Böckle, and F. van der Linden, *SPL Engineering - Foundations, Principles, and Techniques*. Springer IXXVI, 2005.

[4] I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, and J. Bettin. *Domain Engineering. Product Lines, Languages and Conceptual Models*, Springer, 2013.

[5] ISO/IEC NP 26550, *Software and Systems Engineering – Reference Model for Software and Systems PL*, ISO/IEC JTC1/SC7 WG4, 2013.

[6] E. Berard, *Essays in OO Software Engineering*, Prentice Hall, N.Y., 1992.

[7] P. Abrahamsson and P. Keynote: *Mobile Software Development - The Business Opportunity of Today*, in proceedings of the International Conference on Software Development, pp. 20-23. Reykjavik, Island, 2005.

[8] A. Wasserman, *Software Engineering Issues for Mobile Application Developmen*t, FoSER'10, FSE/SDP on Future of Software Engineering Research, pp. 397-400, 2010.

[9] ISO/IEC 25010, *Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models*, ISO/IEC JTC1/SC7/WG6, 2011.

[10] F. Losavio, O. Ordaz, and V. Esteller, *Refactoring-Based Design of Reference Architecture*, RACCIS, vol. 5, no. 1, pp. 32-48.

[11] World Wide Web Consortium, *Mobile Web Application Best Practices*, W3C Working Draft, http://www.w3.org/TR/mwabp, 2010.

[12] Agile Alliance, *Agile Software Development Manifesto.* Retrieved from Manifesto for Agile Software Develop. http://agilemanifesto.org, 2001.

[13] B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, 2003.

[14] A. Spataru, *Agile Development Methods for Mobile App*, MSc. These, University of Edinburg, Edinburg, UK, 2010.

[15] B. Kaleel and S. Harishankar, *Applying Agile Methodology in Mobile Software Engineering: Android Application Development and its Challenges*, Reyrson University, Reyrson, Canada, CSTR. Paper 4, http://digitalcommons.ryerson.ca/compsci_techrpts/4, 2013.

[16] M. Shaw and D. Garlan, *Software Architecture. Perspectives of an Emerging Discipline*, Prentice-Hall, 1996.

[17] J. Herrera, F. Losavio, O. Ordaz, and J. Bøegh, *Product Lines Scoping Using ISO/IEC 26550 Reference Model Considering Software Quality*, ASSIT2016, Bangkok, Thailand, pp. 194-200, indexed by CPCI-SSH, ISBN 9791605953885, http://www.ichss2016.com, July 2016.

[18] I. Mansanet, J. Fons, I. Torres, and V. Pelechano, *GeMMINi: Prototipo de IU Sobre Multiples Dispositivos. Una Estrategia basadas en LPS y MDD*, FAZ, 2011.

[19] K. Czarnecki, C. Hwan, P. Kim, and K. Trygve, *Feature Models are Views on Ontologies*, SPIC, 2006.

[20] T. Gruber, *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*. Technical Report KSL 93-04, KS Laboratory, Stanford University, CA, USA, 1993.

[21] L. Bass, M. Klein, and F. Bachmann, *Quality Attribute Design Primitives and the ADD Method*, SEI White Paper, http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=29604, 2001.

[22] H. Koziolek, R. Weiss, and J. Doppelhamer, *Evolving Industrial Software Architectures into a Software Product Line: A Case Study*. LNCS 5581, pp. 177-193, 2009.

[23] F. Losavio, O. Ordaz, N. Levy, and A. Baiotto, *Refactoring Process for Software Product Lines Design*, (JDLP, pp. 47-58, Lille, France, http://jldp.org/2012/images/jldp2012-actes.pdf, Novembre 2012.

[24] J. C. Herrera, F. Losavio, and O. Ordaz, *QuaDRA: Quality-oriented Design of Reference Architecture for Software Product Lines Based on ISO/IEC 26550*, RACCIS Vol. 6, No.1, pp. 20-38, Enero-junio, ISSN 2248-7441, http://www.fundacioniai.org/raccis, 2016.

[25] L. Chung, B. Nixon, and E. Yu, *Using NFR to Systematically Select Among Alternatives in Architectural Design*, 1st International Workshop on Architectures for Software Systems, pp.31-42, Seattle, USA, 1995.

[26] D. Bjørner, *Software Engineering 3: Domains, Requirements, and Software Design*, Texts in Theoretical Computer Science, Springer-Verlag, Berlin Heidelberg, Germany, 2006.

[27] F. Losavio, O. Ordaz, and I. Santos, *Proceso de Análisis del Dominio Agil de Sistemas Integrados de Salud en un Contexto Venezolano*, ENL@CE, vol. 12, no. 1, pp. 101-134, ISSN 1690-7515, 2015, http://www.produccioncientifica.luz.edu.ve/index.php/enlace/index.

[28] C. Salinesi, R. Mazo, O. Djebbi, and D. Dia, *Constraints: The Core of Product Line Engineering*, RCIS'11, pp 1-10, Gosier, Guadalupe, 2011.

[29] Object Management (OMG), *Unified Modelling Language Superstructure, Version 2.0* (formal/05-07-04), August 2005.

[30] F. Losavio, O. Ordaz, and S. Jean, *Ontological Approach to Derive Product Configurations from a Software Product Line Reference Architecture*, Revista CyT, UP, Argentina, no. 16, pp. 91-127, ISSN 1850-0870, May 2016, http://www.palermo.edu/ingenieria/pdf2016/CyT_16_07.pdf.

[31] L. M. Northrop and P. C. Clements with collaborators F. Bachmann, J. Bergey, G. Chastek, S. Cohen, P. Donohoe, L. Jones, R. Krut, R. Little, J. McGregor, and L. O'Brien, *Framework for Software Product Line Practice*, Version 5.0. SEI, Carneige Mellon University, December 2012.

[32] P. Krutchen, *Architectural Blueprints — The "4+1" View Model of Software Architec*ture, IEEE Software, vol. 12, no. 6, pp. 42-50, November 1999.