Reference Architecture Design: a Practical Approach

Mustapha Derras[1], Laurent Deruelle[1], Jean Michel Douin[2], Nicole Levy[2], Francisca Losavio[3],Yann Pollet[2] and Valérie Reiner[1]
[1] *Berger-Levrault, Boulogne Billancourt, France*
[2] *CNAM – CEDRIC, Paris France*
[3] *Universidad Central de Venezuela, Caracas Venezuela*
{mustapha.derras, laurent.deruelle, valerie.reiner}@ berger-levrault.com, {jean-michel.douin, nicole.levy, yann.pollet}@cnam.fr, francislosavio@gmail.com

Keywords: Software Product Lines (SPL), Software Architecture, Reference Architecture Design, Software Product Quality, Bottom-up Strategy.

Abstract: Reference Architectures (RA) in a Software Product Line (SPL) context are generic schemas that can be instantiated to configure concrete architectures for particular software systems or products of the SPL family. SPLs claim to be reusable industrial solutions to reduce development cost and time to market; however their development requires a huge effort, since the RA must be evolutionary. The goal of this work in progress is to present a practical RA domain engineering method for the Human Resources domain based on a bottom-up strategy applied to the early Scope phase of SPL Engineering. A usual industrial practice in this development is to start from a single existing product built by the enterprise and incrementally derive the RA by adding new elements. Four architectural configurations are developed and quality properties are considered early as major responsible of the SPL variability. Our approach is applied to a real industrial case study.

1 INTRODUCTION

The *architecture* is an abstract representation of a software system with related (connected or communicating) components. (Shaw, Garlan, 1996) (Taylor, Medvidovic, Dashofy, 2009). A component describes a part of the system holding an individual behaviour. The whole system behaviour is defined by the composition of the individual behaviours of the components. Every component must be specifically implemented and it holds certain quality properties (maintainability, usability, security, etc.); they must be identified and more or less satisfied in order that the component can maintain a suitable functional behaviour. The architecture is composed by abstract components in the sense that the implementation details are not known already. An abstract component can be implemented in different ways, but all the implementations satisfy the behaviour of the abstract component. A *concrete architecture* is composed by precise implementations of abstract components.

A *Reference Architecture* (*RA*) (Bosch, 2000), (Clements et al., 2001), (Oquendo et al., 2014), (ISO/IEC 26550, 2015) represents a family of abstract architectures. It highlights the presence of established common components, and of other components that are not common, which are called *variants* and that are grouped into variation points (Pohl et al., 2005). A *variation point* is a particular component defining the variants' common behaviour and their connections with all other architectural components; it identifies a set of variants (eventually empty at first), from which only one can be chosen. RA must satisfy a set of rules describing the constraints that should be satisfied while selecting a configuration with the chosen variants. The goal of RA is to be configured to obtain a particular concrete architecture of a software system by choosing convenient variants. During the configuration of RA, consistency properties between the selected variants must be satisfied. The RA is the main asset of the *Domain Engineering (DE)* lifecycle of *SPL Engineering (SPLE)* (Pohl et al., 2005) (ISO/IEC 26550, 2015).

Our research questions are: "how to develop a RA from existing products or systems" and "how to concretise the RA abstract components into executable modules".

Feature models are frequently associated to RA. A *feature model* (Kang et al., 1990), where every

node of the feature model tree is a variation point and the sub-nodes are the variants, can be associated to RA. The advantage of the feature model, which provides a tree-like view of RA, is the visualization of the choices to obtain the architecture of the system. However, as RA contains variations points, feature models do not provide any new information. Let us note that it is difficult in general to include properties that are not directly perceived by the user, such as the quality properties (Benavides D. et al., 2007). Then, in the present approach we consider building RA without using a feature model.

The main goal of this work in progress is to propose a method to build a RA, inspired in the work of (Losavio, Ordaz, Levy, 2014), and the standard reference model guidelines for SPLE (ISO/IEC 26550, 2015). The method is applied to an industrial case study, the SEDIT Human Resources (HR) System of the French Berger-Levrault Group, based on the one hand, on the behaviour of the abstract functional components representing SEDIT behaviour and on the other hand, considering non-functional requirements. We have started the work on this research topic a couple of years ago with the Healthcare Information Systems Domain (Losavio, Ordaz, Esteller, 2015). The RA is developed following a reactive bottom-up strategy (Apel et al., 2013), meaning that the RA will be constructed from a unique product already built by the industrial partner, thus reducing the whole RA design effort. This work considers the experience with the *Vacation Request Subsystem (VRS)*, which is a major complex functionality defining services offered to municipal communities by the SEDIT system.

Functional Requirements (FR) are the system basic functionalities accomplishing prescribed functional goals, expressed in our case by the enterprise business processes. However, non-functional goals that are required by functionalities to behave properly, i.e., to be functionally suitable (ISO/IEC 25010, 2011), must also be considered. *Non-functional Requirements (NFR)* must be taken into account while studying existing products. There is a general agreement on the fact that NFR must be considered early in the software development lifecycle to increase software maintainability. The SPLE reference model claims that NFR are greatly responsible of RA variability. However, these aspects are considered only at the late domain design stage when the RA is already built, contradicting this assumption. In usual industrial practice these issues are poorly treated, being often left to the responsibility of the particular development platform used. Notice that SPLE favours a global top-down design to build the SPL from scratch considering a very huge and time consuming domain analysis; however, the first SPL scope phase recommends bottom-up strategies since domain existing products have to be analysed. We start using this strategy that also contributes to the light-weightiness of our methodology, reducing the effort in the subsequent DE phases, namely *Domain Requirements Engineering (DRE)* and *Domain Design (DD)*.

The structure of this paper, besides this introduction, is the following: section 2 details the method and the underlying process; section 3 presents the application of the process to the VRS case study; section 4 analyses related works, and finally section 5 provides the conclusion and some perspectives.

2 METHOD TO BUILD RA

SPLE includes two lifecycles: DE where domain knowledge is captured to build RA, and *Application Engineering (AE)* where the concrete products of the SPL family are configured according to the variability model contained in the RA. The proposed method is framed into the SPLE Domain Engineering lifecycle and follows the systematic process shown in Figure 1; it concerns a four-phases process; the first two steps can be used to develop any quality driven single system architecture; the third step defines the Domain Requirement Engineering Architecture, and the fourth step defines the RA, maintaining quality traceability through functional and non-functional components. Figure 1 presents the process as a UML Activity Diagram.

3 APPLICATION OF THE PROCESS

The phases of the process shown in Figure 1 will be detailed in the following sections; however, note that Phase 0.1 was performed for only one product and will not be discussed here.

3.1 Phase 0.2, Activities 0.2.1. 0.2.2, 0.2.3 Identification of Domain Style and Quality Properties

VRS belongs to the family of HR Information Systems (HRIS) that combines a number of systems and processes to ensure the easy management of business employees and data. The programming of data processing systems evolved into standardized routines and packages of Enterprise Resource Planning (ERP) software. Another important issue that HRIS should support is the changing of legal

requirements, such as laws and country specific regulations, for example the SAP Company has to handle manually these issues.

We consider a *business goal* as an observable and measurable end-result having one or more objectives to be achieved within a more or less fixed timeframe. The main business goal of the company is HR and financial management on French local authorities, i.e., communities, departments, regions, etc., where people invested in public offices exercise authority over their territory. The hierarchy of assigned authority is relevant to provide the correct signature to authorize different administrative transactions. SEDIT verifies the HRIS domain hybrid architectural style (Shaw et al., 1996): event-based, layers and client/server model for communication and main quality properties.

One of its most important qualities, the functional suitability (appropriateness), is to be compliant at any time with laws that are constantly changing; the system must guarantee this compliance. The hierarchy of people in office can also change at any time, impacting signature circuits, and this compliance must be also assured. The VRS system studied is a classic Web-based information system. The availability of services is conditioned also by the network connection.

3.2 Phase 0.2, Activities 0.2.4, 0.2.5 - Specify Business Processes and Criteria to Conform Architectural Components

The stakeholders of the company were interviewed to capture the information about the functional behaviour of the system. The workflows relative to the participant (person or system) activities were specified using BPMN; they will not be shown here to abridge the presentation. The idea is to "translate" into architectural components these activities or tasks expressing functional behaviours to accomplish a specific goal. Some automatic tools are available to "translate" business processes into choreography/orchestration of services, requiring in general optimization processes; but there are few automatic tools to translate business processes activities into functional architectural components, such as ArchiMate (The Open Group Architecture Framework) (Open Group, 2012).

The whole VRS representing a service to be accomplished by the SEDIT HRIS, is one of the enterprise business goals.

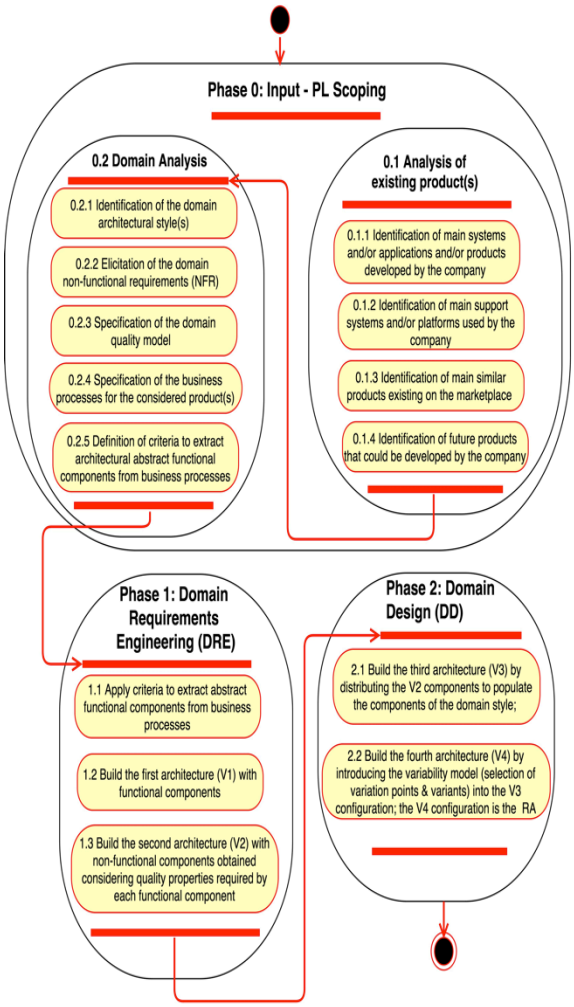


Figure 1. Domain Engineering process to build the RA.

3.3 Phase 1 (DRE), Activities 1.1, 1.2 - First Architecture (V1) with Main Functional Components

The architecture shown in Figure 2 is derived from the BPMN workflows for the VRS, applying the rules shown in Table 1.

The idea is to organize this first configuration by introducing components by grouping the functional activities accomplishing business goals in the stakeholders' lanes; all components are considered at a high abstraction level. Architectural configurations are represented as UML 2.0 logic views (Krutchen, 1995); note that the UML notation we used here does not show components' interfaces. Components are associated to each BPMN stakeholder's lane, to accomplish business goals and sub-goals, in the following way:

- A component can group lane's activities (performed from a human or a system), representing a sub-goal.
- Only functionalities and their cooperation are concerned.

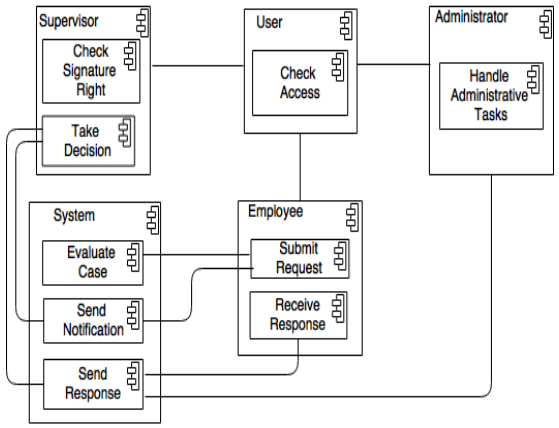


Figure 2. First architecture with functionalities (V1).

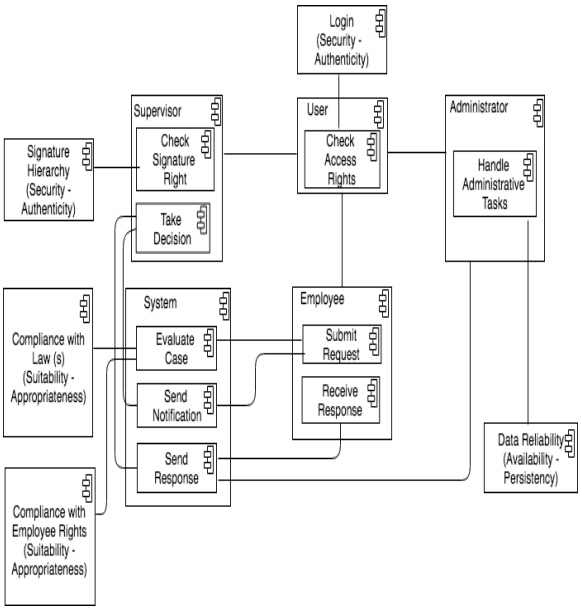


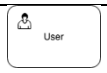
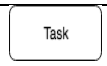
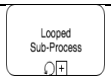
Figure 3. Second architecture with functionalities and quality properties (V2).

3.4 Phase 1 (DRE), Activity 1.3 - Second Architecture (V2) with Functional and Non-Functional Components

Non-functional components represent properties or features not directly perceived by the user (see Figure 3). However, they are required by the functional components to satisfy completely their goals. Notice that the quality properties treated are part of the Domain Quality Model (ISO/IEC 25010, 2011). The functional components of V2 must satisfy four priority quality properties:

- *Security (Authenticity)* realized by non-functional components *Login-Security (authenticity)* to provide access to the right kind of user, and *Signature Hierarchy (authenticity)*, also required by functional component *Check Signature Rights*, to follow the enterprise hierarchy of assigned authority.
- *Suitability (Appropriateness*) is realized by non-functional components *Compliance with Law (s)* and *Compliance with Employee Rights*; this quality property appears because the employee must fill up the right formulary according to legal requirements and must also be compliant with the employee legal rights.
- *Appropriateness* is required by the *Evaluate Case* functional component.
- *Data Reliability* is added as a new non-functional component to satisfy *Availability-Persistency, s*ince the *Administrative Tasks* functional component requires storing results of the whole vacation request process.

Table 1. Rules to convert BPMN workflows into architectural components.

Tasks	*BPMN graphical representation*	*Rules for BPMN conversion to architectural components*
User (human)		A user task is represented as 1 functional component involving an interface
Service		A service task is represented as 1 system functional component
Receive		They are gateways to catch or receive messages; they are represented as connections between the actor or system components
Send		
Abstract		It represents an abstract component that will be specified later on
Looped Sub-process		A looped sub-process is represented as 1 functional component involving an iterative process
Sub-process		A sub-process is represented as 1 functional component

Notice that *Usability* and *Efficiency* are priority quality properties for the *User Interface (UI)*, but they will not be considered here because they depend on

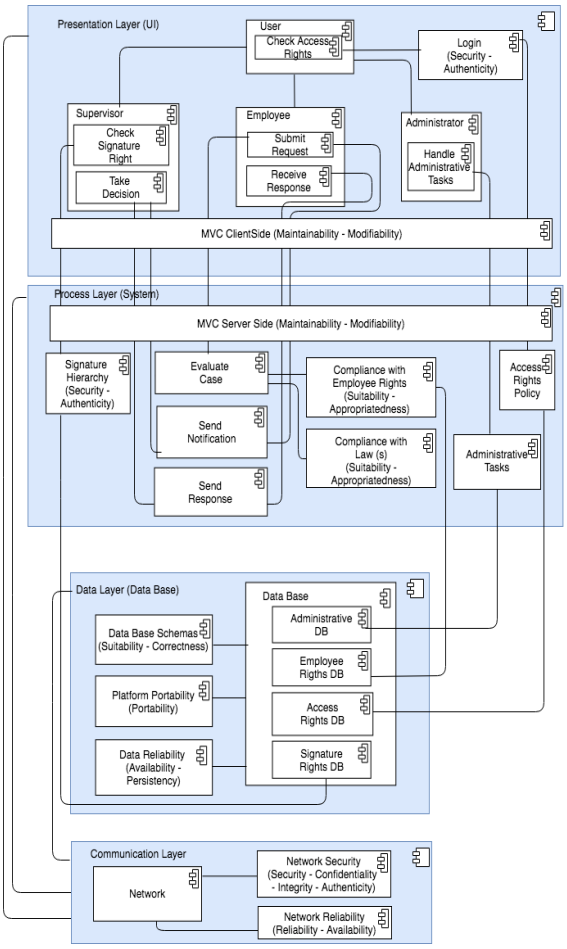


Figure 4: Third architecture with domain style (V3).

the UI particular design and not on the system architecture. The *Maintainability (Modifiability)* quality property has not yet been satisfied in the V2 configuration, and it will appear when the architectural style is considered.

3.5 Phase 2 (DD), Activity 2.1, Third Architecture (V3) with Domain Style

The choice of an architectural style depends on the domain. Each V2 component has been placed in the V3 configuration as a sub-component of the components of the style (see Figure 4). Notice that some components are divided into several sub-components in order to satisfy the style; that is why a *Data Base* component is introduced to populate the Data Layer.

Each layer is considered as a component; hence we have *UI* as *Presentation Layer*, *System* as *Process Layer*, *Data Base* as *Data Layer*; finally the *Communication Layer* communicates the different layers and exterior systems/services. Notice that the UI requirement *Maintainability (Modifiability)* is usually solved by the MVC (Model, View Controller) pattern, and it is split into two components *MVC-ClientSide* and *MVC-ServerSide,* to interface Presentation and Process layers (see Figure 4). The *Process Layer* corresponds to the VRS business goal and it contains all the remaining functional and non-functional components of the V2 configuration with the same connections. Component *Access Rights Policy* is introduced to solve security for access control, and *Signature Hierarchy* to satisfy the *authenticity* imposed by the signature rights. *Administrative Tasks* is in charge of managing and maintaining the vacation request information and a database is introduced to satisfy persistency. In the Data Base Layer four functional sub-components have been added to satisfy NFR requirements, *Administrative DB, Employee Rights DB, Access Rights DB,* and *Signature Rights DB*. *Correctness* is required by data base schemas, *persistency* to store/get the process results and *portability* since they are presumably relational databases and do not work directly with object-oriented languages like Java. In the Communication Layer the UI can connect to the *Network* for e-mail and other external services through a usual browser and to communicate with the Process Layer; *Security (authenticity, confidentiality, integrity)* in message transmission should be realized by *Network Security,* and *Reliability (Availability*) for system access is realized by *Network Reliability*; they are also priority quality properties that have been added as non-functional components to the Communication Layer since they can jeopardize the whole Web-based system functioning.

Notice that this third step could have been applied at the beginning of the process since the domain style is known; however we have decided to apply it here to ease repeatability with small incremental steps.

3.6 Phase 2 (DD), Activity 2.2 - Fourth Architecture (V4) with Variability Model, the RA

Software variability is the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context (Van Grup, 2000). In SPL the term variability model (Pohl et al, 2005) is used to organize variant elements in such a way that they can be reused at the moment of deriving concrete products from the RA. Variation Points are elements (placeholders) of the variability model; in

our case they may be functional or non-functional components and they are denoted here by <<name>> as UML stereotypes (see Figure 5). A variation point denotes a set of components called *variants*; the connector of a variation point denotes a set of connectors, each one belonging to a variant.

The activity 2.2 of our process is the identification of the variation points and their variants from the architecture obtained in V3. In order to achieve this, other similar existing products should have been studied or future products to be built should be considered to cope with the SPL evolution, and new planned functionalities could also have been added, see activities 0.1.3 and 0.1.4 of Phase 0.1. If this study is not done, variation points and their variants are identified directly on the V3 architecture, establishing some criteria, leading to the configuration shown in Figure 5, the RA for a VRS family of systems. All RA abstract functional and non-functional components should be traced and concretized to the corresponding reusable code modules; if no reusable module can be found, it should be adapted or constructed to provide all assets to perform the concrete product derivation from the RA and guarantee complete functional suitability. The Variability Model in the V4 last configuration is defined by the following specific activities:

a. Describe the *Contex*t as an external system, to proceed with the Application Engineering lifecycle of the SPLE model; the RA variation points should be instantiated to derive the SPL concrete products. This system is in charge of putting into operation or "implement" a concrete product on the client demand, according to a configuration and/or customization process. This part is still an on-going work.

b. Identify functional and non-functional components that can have different choices to accomplish their goals; mark them as stereotyped variation point components.

c. For each variation point, identify possible variants, such as for example the Hibernate tool to solve Oracle portability to Java objects.

d. Describe the dependencies and constraints among choices of variants.

4 RELATED WORKS

Many works are found in the literature about SPL development and industrial adoption, but few treat the whole SPLE process. In this work we are more concerned with the Domain Requirements life cycle of SPLE, where the RA is built. Three literature reviews will be discussed.

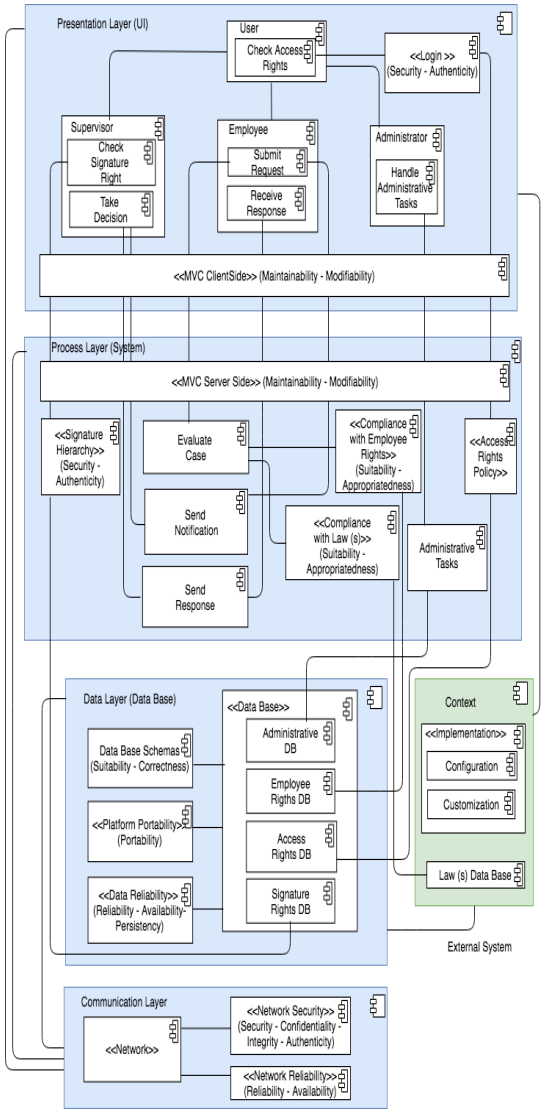


Figure 5: Fourth architecture with Variability Model

The work of (Chen et al., 2010) treats variability management for the RA evolution; this topic has been studied for almost 20 years. An empirical study using focus group of SPL experts in different fields (industry, consultancy, academy, etc.) were questioned to collect data; two of the first contributions to variability management research and practice were the known Feature-Oriented Domain Analysis (FODA) (Kang et al., 1990) method and the Synthesis approach (Kasunic, 1992) focused on a family of similar systems for a business area. This approach arises for SPL from evidence that, within a class of systems, the understanding of similarities provides significant leverage for constructing a great variety of high-quality systems cheaply and reliably. With Synthesis, a domain is conceived as a realization

of the declared business objectives of the specific organization. Business goals determine the types of systems to build and customers. These goals are set on the expertise already available within the organization, as a result of experience in building systems in the past. Among the technical issues, many challenges were considered, such as: - Requirements: complexity management in feature modelling that affects maintainability, the lack of standards in feature representations that affects understandability, and the extraction of commonality and variability from existing poor documentation. - Architecture design: extract the variability from technical artefacts of different similar products and build a common architecture for those products. Decisions are not documented properly and are hard to be retraced. - Evolution: when requirements change, in some cases, the existing architecture does not support the required variability in the new requirements. In an extreme case where the SPL started with one single product, evolving the architecture towards an SPL was seen as an issue. With respect to non-technical aspects, several issues were considered, such as the Business Model that does not encourage reuse. In conclusion, it was found that variability management relies mainly on the technical aspects, the standardization of variability representation and modelling, the documentation, and management.

In our work, the use of standards is favoured; we adopt a bottom-up strategy to build the RA from a single product, following the mentioned Synthesis approach, starting from a BPMN specification of business processes, extracted from interviewing industrial stakeholders.

The work of (Mazo el al., 2014) on the ERP domain aims to identify and analyse the different ways to improve ERP engineering issues with methods, techniques and tools provided by SPLE. An ERP system is not a family of applications, but a single application. However, just like a product line, configuration mechanisms are used to satisfy the various requirements from different companies. ERP implementation (customization and configuration) focuses the major research axis in ERP. The goal is to take into account the specific needs of the organization when ERP standards and configurable features cannot achieve them. Variability management and the ability for the system to be configured/customized and adapted to a potentially undefined number of environments are two main challenges. ERP *configuration* deals with parameter changes and it can take several months and no results can be guaranteed; *customization* instead refers to interface development or code modification, and it requires to be regularly updated to maintain the system agility. This complexity of ERP systems is maybe the most important obstacle to use ERP systems in an efficient and predictable way. The paradigm was adopted to configure and/or customize ERP systems using different methods with varying approaches. However, from the point of view of ERP customers and users, it is difficult to differentiate between product variability and customization.

With respect to our work, the compliance with laws has been found as a non-solved problem; the configuration of the legal requirements as reusable assets is still an open problem in the HRIS domain; this issue must be handled by the SPLE Application Engineering lifecycle.

The ERP domain and SPLE approach are also treated in (Ouali, 2011), focusing on the evolution issue and on the problem of tool support for these approaches. SPL fits to ERP business, and ERP systems can benefit greatly from the concepts of commonality and variability to enhance maintainability or evolution capability. Four methods are reviewed, Van Group, Ziadi, Delstra and Djebbi, being the Main drawbacks: the lack of sufficient tool support and poor interactivity with their users. The SPL development approaches themselves are not enough automated for deriving automatically a concrete product from the RA; most of them use proprietary notations which can handle problems of standardization and interoperability; they do not cover all aspects of SPLE. Every method tries to focus on a particular part of the SPL construction process. However, in this work the discussion on the experimentation with ERP is very limited, and no case study is provided.

The Office of Management and Budget (OMB, 2012) launched the Human Resources Line of Business (HR LOB) Data Model, an effort to build a set of enterprise architectures in compliance with the USA Federal Enterprise Architecture standards for the human resources business function. It is an interesting and complete work and conceptual elements can be used for the general HRIS domain, however it is data-oriented for enterprise architectures and does not concern directly the RA development for SPL.

The revised works mention standardization as an open problem; the ISO/IEC 26550 standard has been recently formulated to provide a framework for SPLE guidelines (ISO/IEC 26550, 2015). The problem of early consideration of software quality is discussed affirming its importance, but it is left to the late Domain Design phase. The bottom-up strategy is recommended in the SPLE Scoping phase to reduce the effort in the subsequent development phases. In our present work we adopt this approach. A couple of years ago we started to work on a SPL Domain

Engineering quality-driven method based on the extractive bottom-up strategy, considering several existing products in the Healthcare Information Systems domain (Losavio, Ordaz, Levy, 2014), (Losavio, Ordaz, Esteller, 2015).

5 CONCLUSIONS

The main outcome of this paper is the definition of a domain engineering method to build incrementally a RA, according to a bottom-up reactive strategy. This method, inspired in (ISO/IEC 25010, 2011), defines a systematic and repeatable practical process to build the RA, and it has been applied to a real industrial case. In addition, we have considered the compliance (functional suitability-appropriateness) to legal requirements (laws and regulations) as a priority quality requirement, since they change often overtime and their management is a time consuming and non-systematic process in HR systems involving full-time human resources. On the other hand, the traceability among functional, non-functional components and their technical solutions has been clearly established by taking into account quality requirements, early in the SPLE lifecycle.

Among the perspectives, it is clear that more lower level architectural configurations can be produced from Figure 4 to detail the variants of variation points, such as 'e-mail' and 'electronic document' in <<Submit Request>>. However, to solve our second research question, an intermediate layer has to be defined to link the abstract RA components to the reusable modules of code, which are all reusable SPL assets. Moreover, mechanisms based on architectural patterns are under design to determine the choice of variants and the addition of new functionalities, without modifying the existing code. Our present work is a first step towards the idea of offering a "product on demand" for HR management to French territorial communities, by facilitating a configuration/ customization of on-line process of the HR system.

REFERENCES

Apel, S., Don, S., Batory, S., Kastner, C., Saake, G., 2013. Feature-Oriented Software Product Lines - Concepts and Implementation. Springer.

Benavides D. et al., 2007. FAMA: Tooling a Framework for the Automated Analysis of Feature Models. In *Workshop on Variability Modelling of Software-intensive Systems*.

Bosch J., 2000. Design and Use of Software Architectures – Adopting and evolving a product-line approach, Addison-Wesley.

Chen, L. and Babar, M. A., 2010. Variability Management in SPL: An Investigation of Contemporary Industrial Challenges, *SPLC 2010*, *LNCS 6287, 166–180*, Springer-Verlag Berlin Heidelberg.

Clements, P. and Northrop, L., 2001, SPL: practices and patterns, 3rd ed. Readings, MA, Addison Wesley,

Van Grup J., 2000. Variability in Software Systems, the key to software reuse. Sweden: Univ. of Groningem.

ISO/IEC 25010, 2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models. ISO/IEC JTC1/ SC7/ WG6, Draft.

ISO/IEC 26550, 2015. Software and Systems Engineering – Reference Model for Software and Systems Product Lines, ISO/IEC JTC1/SC7 WG4.

Kang, K.C., et al. 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. *Tech. Report CMU/SEI-90-TR-021, SEI*.

Kasunic, M., 1992. Synthesis: A reuse-based software development methodology. Process Guide, Version 1.0. *Tech. Rep.,* Software Productivity Consortium Services Corporation.

Krutchen, P., 1995. Architectural Blueprints — The "4+1" View Model of Software Architecture, *IEEE Software* 12 (6) 42-50, November.

Mazo, R., Assar, S., Salinesi, C. and Hassen, N. B., 2014. Using Software Product Line to improve ERP Engineering: Literature Review and Analysis, *Latin American Journal of Computing LAJC*, Vol. 1 (1).

Ouali, S., Kraiem, N. and Ben Ghezala, H., 2011. *International Journal of Software Engineering & Applications (IJSEA)*, Vol.2, No.2, April.

Pohl, K., Bockle, G. and Van Der Linden, F., 2005. SPL Engineering: Foundations, Principles, and Techniques, Springer.

Shaw, M. and Garlan, D., 1996. Software Architecture. Perspectives of an emerging discipline, Prentice-Hall.

Oquendo et al., 2014. Reference Architectures, Chapter 2, Edited by Mourad C. Oussalah, Wiley.

Supakkul S., Chung L., 2004. Integrating FRs and NFRs: A Use Case and Goal Driven Approach.

Open Group, 2012. ArchiMate 2.1 Specification, www.opengroup.org/subjectareas/enterprise/archimat.

Office of Management and Budget (OMB), 2006, Human Resources Line of Business (HR LOB) Data Model Version 1, Office of the Chief Information Officer, Washington, DC 20415, February, OCIO-2006-01.

Losavio, F., Ordaz, O. and Levy, N., 2014. Refactoring.

Graph for Reference Architecture Design Process. In *AFADL*, 103-108, CNAM, Paris.

Taylor R. N., Medvidovic N., Dashofy E., 2009. Software. Architecture: Foundations, Theory and Practice, Wiley.

Losavio, F., Ordaz, O. and Esteller, V., 2015. Quality-based Bottom-up Design of Reference Architecture Applied to Healthcare Integrated Information Systems, In *9th RCIS, IEEE,76-81*, Athens, Greece.