Towards bridging the gap between domain and application design

Mustapha Derras, Laurent Deruelle
Berger Levrault
Boulogne Billancourt, France
{mustapha.derras, laurent.deruelle}@berger-levrault.com

Nicole Levy
Cedric
CNAM
Paris, France
Nicole.Levy@cnam.fr

Francisca Losavio
Escuela de Computación
Universidad Central de Venezuela
Caracas, Venezuela
francisca.losavio@ciens.ucv.ve

Abstract*—The traceability among different abstraction levels in a software development process is still an unsolved problem. Our present goal is to reduce the gap between the high level abstract software product line (SPL) reference architecture (RA) and the concrete application design, by defining first traceability links between the RA components and the technological assets used by the enterprise requiring the SPL, and then study the architectural components interfaces by adapting the Domain Realization phase guidelines; they are proposed by Böckle, Pohl, and van der Linden (2005), and by the ISO/IEC 26550 (2015) reference model for SPL engineering. This preliminary work establishes links between the RA and an external configuration system to facilitate the compliance with laws, which has been found as a major problem while configuring Human Resources (HR) systems. Our approach is illustrated with an industrial case study, the Vacation Request subsystem of the SEDIT HR system of the Berger-Levrault enterprise, widely used in French and foreign communities.

***Keywords*—software product line, reference architecture, domain and realization design, application design**

I. Introduction

The correspondence between two architectural views [1] of a software system has been widely discussed in the literature, however it still remains a problem. *A Reference Architecture (RA)* for a *Software Product Line (SPL)* [2], is built in the *Domain Design* phase as the main asset of the *Domain Engineering (DE)* lifecycle; the *SPL Engineering (SPLE)* Model of [3], now incorporated into the new ISO/IEC 26550 [4] is followed, see Fig. 1. The main idea of this work is to refine the RA variability model before starting the Application Design, using the *Domain Realization* phase of DE. The mapping between the architectural logic view and the physical or deployment view will be discussed as a first step, by specifying the technology related to each architectural component; new components may be introduced at this stage. Then the architectural component interfaces will be defined. We will explore if the gap between different architectural abstraction levels, the Domain Design and the Application Design, can be reduced. The Domain Realization will be used as an intermediate abstraction level, to facilitate the configuration step in the *Application Engineering (AE)* lifecycle. Configuration is unavoidable in SPL during the Application Design phase to produce concrete applications derived from the RA core assets and the variability model; for example, the systems migrating to a cloud structure need to be configured to be used by different clients; the configuration consists in setting the convenient variants. Note that businesses are coping with the challenge of upgrading and optimizing enterprise applications performing configuration using native tools, without modifying the system code, and avoid a customization process that requires code manipulation [15]. Vendors are providing more options to configure applications for particular needs and industries. But in some cases, configuration options are getting so numerous and layered that they present challenges of their own. Any way, a convenient instance of the RA has to be configured to obtain a concrete software application or product of the SPL family, responding to client requirements. This step is still on the threshold between DE and AE. The problem of relating the abstract components of the SPL RA [2], [3] with the concrete components of a software product of the SPL family using the right reusable assets, is not completely solved. The reusable assets offered by the actual technology (toolkits, APIs, tools, etc.) used in industrial developments, solve nowadays aspects that had to be carefully programmed before. Some examples are the user interface quick development including the MVC maintainability concern often solved by Angular Js/RESTful, and the portability of the relational database to objects with the Hibernate tools. However, it is claimed that the architecture is still the key to the success of any software project since it is the first design artefact that begins to place requirements into a solution space. The quality attributes of a system, such as performance, modifiability, and availability should be considered by the architecture; if the architecture is not suitable from the beginning for these qualities, it is very difficult to achieve them by some miracle later [6]. The architecture determines the structure and management of the development project as well as the resulting system, since teams are formed, and resources allocated around architectural components. In a previous work [7], a RA was constructed for the Vacation Request subsystem of the Berger-Levrault SEDIT *Human Resources (HR)* system [8], using a bottom-up strategy with a single product already built by the enterprise. The RA was obtained with a variability model containing mostly non-functional components related to the functional components of the core asset; these non-functional components represent the quality properties to be satisfied by each functional component in order to be compliant with functional suitability [9]. It is important to notice that these non-functional components were introduced to assume the responsibility to check that the quality property will be actually present in the concrete product. All the non-functional components are variation points that implement different solutions according to the technological evolution. Each choice will be linked to a variant attached to a variation point, realizing the traceability. For example, the non-functional component *<<Compliance with law(s)>>* will be directly linked to the rules specifying the law(s).

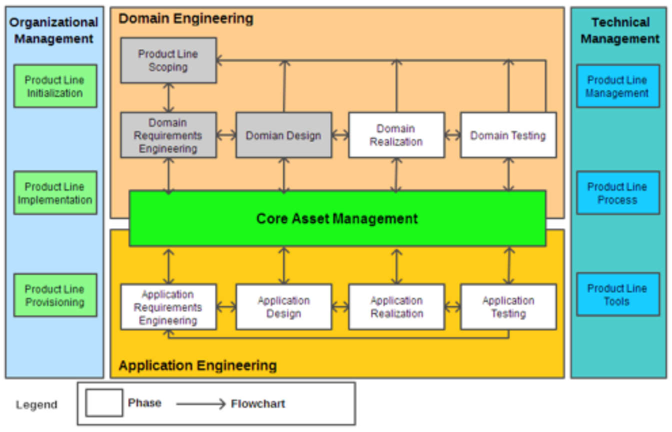

Fig. 1. ISO/IEC 26550 SPLE (Software Product Lines Engineering) Reference Model Framework [4]

Recall that the RA or domain architecture determines the structure and the texture of the concrete applications or products of the SPL. According to [3], the *structure* describes the decomposition that is valid for all applications of the SPL (commonality and variability). The *texture* is the collection of common rules guiding the design and realization of the parts, and how they are combined to form applications. It defines common ways to deal with variability in domain or application design and realization; it consists of coding rules and general mechanisms such as styles [10] and design patterns [11] to deal with specific situations/solutions that may occur during design, realization, and coding. The Domain Realization phase is focused on building this architectural texture that will be used to design the concrete applications of the SPL family.

A process was defined in [7], following the first three phases of the DE lifecycle to obtain the RA, shown in grey in Fig.1. Our present goal is to reduce the gap between the high-level abstract RA and the more concrete application design by defining first traceability links between the RA components and the reusable technological assets. In order to do this, the SPLE Domain Realization phase guidelines [3], [4] will be adapted to deal with a more detailed design concerning the reusable software components and their variants.

The RA represented in UML 2.0 in Fig. 2 is a logic view of the architecture, incorporating elements of the process view, such as the layered domain style used by HR systems. According to [1], architectural views can be combined. However, in this UML 2.0 notation, components interfaces are not shown, nor the provide/require in the connections; the components, which are variation points are denoted as stereotypes by << *component name* >>. Components that are not variation points are considered common components that will be present in all the products of the SPL family.

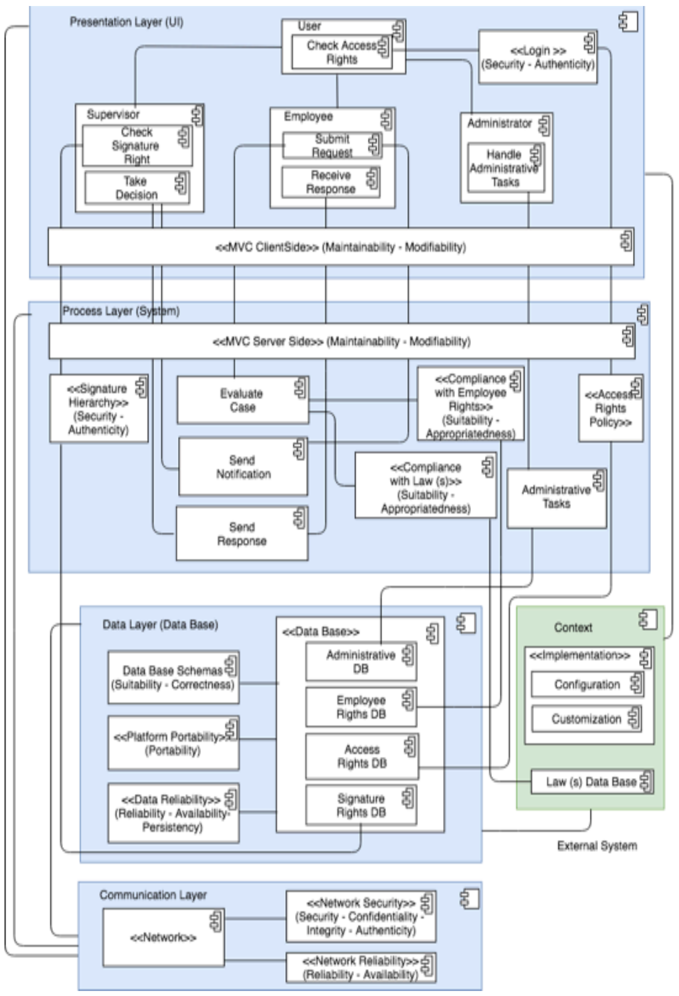
Presentation Layer (UI)
User
Check Access Rights
<<Login >> (Security - Authenticity)
Supervisor
Check Signature Right
Take Decision
Employee
Submit Request
Receive Response
Administrator
Handle Administrative Tasks
<<MVC ClientSide>> (Maintainability - Modifiability)
Process Layer (System)
<<MVC Server Side>> (Maintainability - Modifiability)
<<Signature Hierarchy>> (Security - Authenticity)
Evaluate Case
<<Compliance with Employee Rights>> (Suitability - Appropriatedness)
<<Access Rights Policy>>
Send Notification
<<Compliance with Law (s)>> (Suitability - Appropriatedness)
Administrative Tasks
Send Response
Data Layer (Data Base)
<<Data Base>>
Administrative DB
Data Base Schemas (Suitability - Correctness)
Employee Rigths DB
<<Platform Portability>> (Portability)
Access Rights DB
<<Data Reliability>> (Reliability - Availability-Persistency)
Signature Rights DB
Context
<<Implementation>>
Configuration
Customization
Law (s) Data Base
External System
Communication Layer
<<Network>>
<<Network Security>> (Security - Confidentiality - Integrity - Authenticity)
<<Network Reliability>> (Reliability - Availability)

Fig. 2. Reference Architecture for the Vacation Request subsystem of BL SEDIT [7]

This paper is structured as follows besides this introduction: the second section describes the Domain Realization phase: the correspondence between architectural components and the technological tools and the definition of the components interface. Guidelines to perform these activities are provided. The third section discusses some related works and finally the conclusion and perspectives are presented in the fourth section.

II. Domain Realization

A. Generalities

Domain Design is the DE process of SPLE in which the commonality and the variability of the SPL are defined to conform to the RA structure. The input for the domain realization sub-process consists of the RA including the list of reusable software artefacts already present or to be developed. Each component should be planned, designed, and realized for reuse in different contexts supported by the component interface. It is important to note that the result of this phase consists of loosely coupled, configurable components, not of a running application. Domain realization can incorporate configuration mechanisms into the components to realize the variability of the SPL. Traceability links between the artefacts of the reusable platform will facilitate systematic and consistent reuse. Our problem concerns the way to establish consistent traceability links between the RA abstraction level and the design level of the concrete product, saying Application Design and Realization (see Fig. 1). The output of Domain Realization encompasses the design and "implementation" aspects of reusable software components, and their interfaces, for example some kind of configuration files. Components realise variability by providing suitable configuration parameters in their interfaces. Notice that Domain Realization differs from the realization of single systems mainly because the result consists of loosely coupled, configurable components, not of a running application.

The industrial case study of the Vacation Request system will illustrate our approach.

B. *Correspondence between architectural components and technical tools*

As a first step to provide a rough process for the Domain Realization phase, we will establish the correspondence between architectural components and the technological tools used by the enterprise, which are known reusable and available assets. Notice that technology changes constantly, then all the mentioned tools are potential variants, considering the SPL variability model approach. In our case, each RA architectural component will correspond to a set of modules, which have been already built by Berger-Levrault. For example, two systems implemented in Java are the main modules of the SEDIT Vacation Request subsystem: - the *X.Net* human resources manager; it allows the configuration of a validation system for a vacation demand to check the hierarchy of the staff responsible of authorizing the request; it corresponds to the RA *<<Signature Hierarchy>>*; - the *e.SEDIT RH* to automatize the staff administrative tasks, favouring also collaborative work; it corresponds to the RA component *Administrative Tasks* [12]. The architectural style for the RA is event-based, layers and follows a client/server model for communication [10], used in the HR domain.

The SEDIT tools of the technological platform, i.e., main frameworks, toolkits, APIs and services used in the development stage [8] are mostly open source and they are shown in Table 1.

Guidelines:

Input: the RA (in our case, the Vacation Request structure of RA in Fig. 2), the list of reusable software artefacts that are already available or to be developed. In the SEDIT case, the list of technological tools is used.

- For each component and sub-component in a layer, the relation with the corresponding technical tool or with an architectural component present in the subsequent layer, are shown.
- Technological (s) solution (s) is (are) listed for each component.

Output: loosely coupled components that can be configured by their interfaces.

Funded by Berger-Levrault

All the connections between layers follow the *REST* system architectural style, where resources are directed only through their URLs, via the http/https protocol. The *JAMon* system monitors all connections between architectural components, including message passing and RMI (Remote Method Invocation). The technical tools/solutions in Table 1 are used to implement the architectural components as reusable modules at Application design level.

TABLE I. CORRESPONDENCE BETWEEN ARCHITECTURAL AND TECHNOLOGICAL COMPONENTS

Architectural components	**Solutions realized with technological or architectural components**
Presentation Layer (UI)	*- Angular Js (client-side), Flying Saucer, RESTful;*
• User	
○ Check Access Rights	- << Access Rights Policy>>;
• <<Login>> (Security-Authenticity)	*- Angular Js (client-side), Flying Saucer, LDAP;*
• Supervisor	*- Angular Js (client-side), Flying Saucer, RESTful, D3JS, KSL;*
○ Check Signature Rights	- <<Signature Hierarchy>>;
○ Take decision	- Send Notification; Send Response;
• Employee	*- Angular Js (client-side), Flying Saucer, RESTful;*
○ Submit Request ○ Receive Response	- Evaluate Case; Send Notification; Send Response;
• Administrator	*- Angular Js (client-side), Flying Saucer, RESTful;*
○ Handle administrative tasks	- Administrative Tasks;
<<MVC Client Side>> (Maintainability – Modularity)	*- Angular Js (client-side), Flying Saucer, RESTful; <<MVC Server-Side>>*
Process Layer	
• <<MVC Server-Side>>	*- Angular Js (server-side), Spring, Log4j, SLF4j;* <<MVC (Client-Side)>>;
• <<Signature Hierarchy>> (Security – Authenticity)	- X.Net, Signature Rights DB, *Spring, Log4j, SLF4j, Spring Security, Kerberos Security,;*
• Evaluate Case	*- Spring, Log4j, SLF4j;*
• Send Notification	*- Spring, Log4j, SLF4j;*
• Send Response	*- Spring, Log4j, SLF4j;*
• Administrative Tasks	- e-SEDIT, *Spring, Log4j, SLF4j;*
• <<Compliance with Employees Rights>> (Suitability - Appropriateness)	- Employees Rights DB;
• <<Compliance with Law(s)>> (Suitability – Appropriateness)	- Law (s) DB;
• <<Access Rights Policy>>	- Access Rights DB, *Spring, Log4j, SLF4j, Spring Security, Kerberos Security;*
• <<Data Access – Portability - Persistency/Availability >>	*- Hibernate, Hibernate 4GWT, Hibernate JPA, Hibernate Envers, CAS;*
Data Layer (Data Base)	
• <<Data Base>> ○ Administrative DB ○ Employee Rights DB ○ Access Rights DB ○ Signature Rights DB	*- Oracle, SqlServer, PostgreSql, Informix, MySql;*
• Data Base Schemas	*- Oracle, SqlServer, PostgreSql,*

(Suitability – Correctness)	*Informix, MySql;*
Communication Layer • *<<Network>>* • <<Network Security>> (Security – Confidentiality – Integrity – Authenticity) <<Network Reliability>> (Reliability – Availability)	 *- LAN, WAN* *- http, https;* *- It depends on the stability of the connection;*
Context External System • <<Implementation>> ○ Configuration ○ Customization Law (s) DB	 *- Spring* *- Set of Rules*

C. *Specification of components' interfaces*

A new version of the Vacation Request RA presented in Fig. 3 in UML 2.0 is built in this second step. Components and their interfaces are shown in terms of required/provided resources [13]. The study of the technology used by the enterprise (see Table 1) has provided some hints to update the abstract logic view of the original RA design (see Fig. 2), for example by adding/eliminating components or sub-components. A new <<*Data Access*>> component has been located in Process Layer; it will be used to interface the Process and Data Layers, to ensure data portability and persistency/availability. *Context* is an external system that will provide information to the <<*Compliance with Law (s)*>> component, through a configuration operation, parameterized by the entity, the staff characteristics, the law text, and a configuration file that captures the specificity of the law's evolution. Notice that one of the main problems found in [3] and [7], was that the variability in HR systems was due mostly to the frequent law changes and not much to changes in the main functionalities. Moreover, this process is usually performed in a quasi-manual way at a quite high cost.

The *Context* external system is not connected to the User Interface (UI) Layer since the user that will operate this system does not belong to the Vacation Request set of users (Employee, Supervisor and Administrator); he will operate the Vacation Request configuration externally, updating the law changes. A solution benefiting for example from the Java language features, such as the injection principle and related design patterns [11], to make changes independently from the coded lower levels modules, could be implemented within the *Configuration* component. It is clear that other configuration solutions could be available, that is why the main component <<*Implementation*>> of the *Context* system is a variant. If the Vacation Request system is offered on the cloud, it will be available and used by each client with the appropriate configuration, once the *Context* system, operated by an external user, has concluded its tasks.

The parameters in the component interfaces could be defined as: *entity = {country, region, city, community. ...}*; *staff-id = {id, password, status, ...}*; *access, notify, sign-approval, rights-check: Boolean*; *period: dates for the vacation request or number of days required and period of the year*; *eval-required: Boolean (push button to activate the evaluation in Process Layer*; *updated-data: results of administrative tasks related with the vacation request of an employee*; *law: data structure representing the law text that can be expressed as a set of rules*; *updated-config: updated configuration file*; *config-file: reusable configuration file*; this information can be retrieved from the *Context* system expressed and executed in a programming language, for example Java, benefiting from the language features, such as injection principle and design patterns [11], to perform the configuration according to the law changes, independently from code.

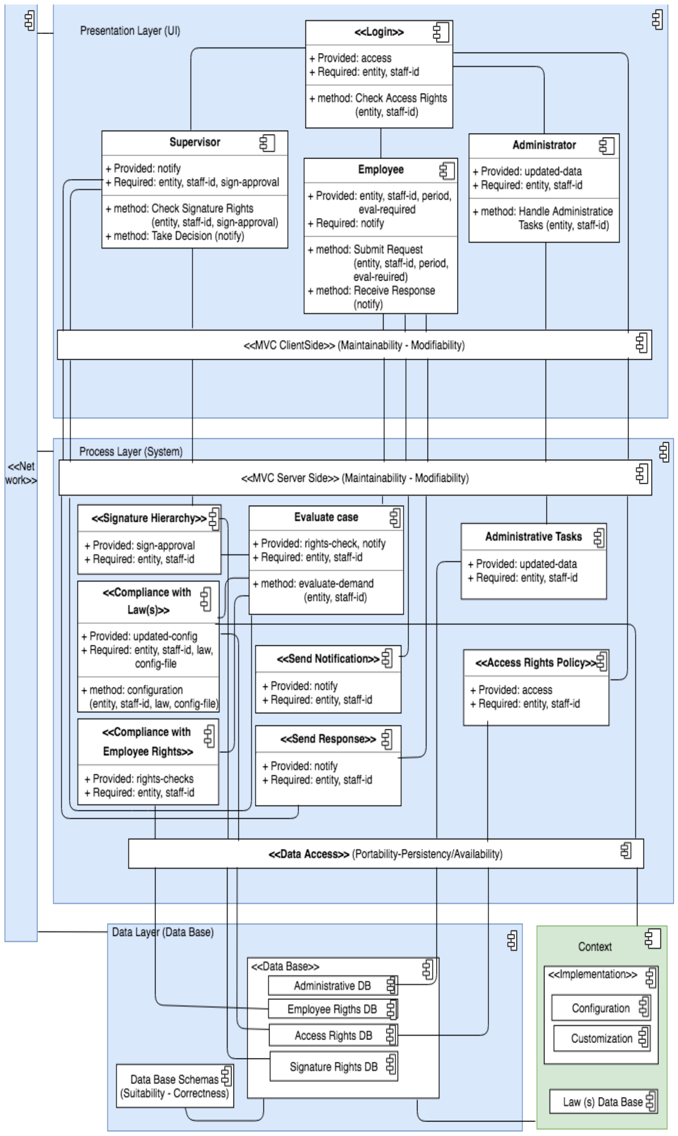
Presentation Layer (UI)
<<Login>>
+ Provided: access
+ Required: entity, staff-id
+ method: Check Access Rights (entity, staff-id)
Supervisor
+ Provided: notify
+ Required: entity, staff-id, sign-approval
+ method: Check Signature Rights (entity, staff-id, sign-approval)
+ method: Take Decision (notify)
Employee
+ Provided: entity, staff-id, period, eval-required
+ Required: notify
+ method: Submit Request (entity, staff-id, period, eval-reuired)
+ method: Receive Response (notify)
Administrator
+ Provided: updated-data
+ Required: entity, staff-id
+ method: Handle Administratice Tasks (entity, staff-id)
<<MVC ClientSide>> (Maintainability - Modifiability)
<<Net work>>
Process Layer (System)
<<MVC Server Side>> (Maintainability - Modifiability)
<<Signature Hierarchy>>
+ Provided: sign-approval
+ Required: entity, staff-id
Evaluate case
+ Provided: rights-check, notify
+ Required: entity, staff-id
+ method: evaluate-demand (entity, staff-id)
Administrative Tasks
+ Provided: updated-data
+ Required: entity, staff-id
<<Compliance with Law(s)>>
+ Provided: updated-config
+ Required: entity, staff-id, law, config-file
+ method: configuration (entity, staff-id, law, config-file)
<<Send Notification>>
+ Provided: notify
+ Required: entity, staff-id
<<Access Rights Policy>>
+ Provided: access
+ Required: entity, staff-id
<<Compliance with Employee Rights>>
+ Provided: rights-checks
+ Required: entity, staff-id
<<Send Response>>
+ Provided: notify
+ Required: entity, staff-id
<<Data Access>> (Portability-Persistency/Availability)
Data Layer (Data Base)
<<Data Base>>
Administrative DB
Employee Rigths DB
Access Rights DB
Signature Rights DB
Data Base Schemas (Suitability - Correctness)
Context
<<Implementation>>
Configuration
Customization
Law (s) Data Base

Fig. 3. The Vacation Request RA with the interfaces of components

Guidelines:

Input: RA components annotated with reusable technological tools (see Table 1);

- For each component, establish the provided/ required resources; new components can be added, and/or existing components can be deleted; specify the corresponding parameters; define the main methods (corresponding to the RA abstract sub-components, see Fig. 2) used by the component.

Output: RA structure expressed in UML 2.0 showing also the RA texture with components' interfaces (see Fig. 3).

Let us note that most of the functional components of the RA are common components; the SEDIT technological platform provides concrete solutions for several variants of the RA variation points which are non-functional components (see Table 1): *<<MVC Client-side>>* with *Angular Js (client-side)*, *<<MVC Server-side>>* with *Angular Js (server-side)*, *<<Login>>* with *LDAP*, *<<Signature hierarchy>>* with *Spring Security, Kerberos Security*, *<<Data Access>>* with *Hibernate*, and *<<Data Base>>* that is actually settled to *Oracle*. As we have already pointed out, *RESTful* and *JAMon* are followed for communications.

The problem remains however with the *<<Compliance with Law(s)>>* variation point for which some solution has to be provided. In this case the *Context* external system could be a solution with the *Configuration* component. The *<<Implementation>>* will consist of using the Java injection facility and laws will be expressed as a set of rules [18]. The design of this system is still an on-going work.

III. RELATED WORKS

Four works are discussed in this section, the first two are dedicated to the SPL development process, and the last two focuses on the HR domain and the migration to the cloud structure, but do not treat the SPL context. In the near future, Berger-Levrault would like to offer a cloud solution for his clients by software product lines for the main functionalities offered by the SEDIT system. Our work will contribute to design a configuration system to treat the problem of laws variation and compliance found in HR systems; as it has been pointed out, these configuration tasks are in general manually solved, requiring much effort and cost.

The paper of Käkölä [14] discusses the projects and future directions for SPL standardization. The new standard ISO/IEC 26550 [4], which was an on-going project, is discussed. It presents a reference model for Software Product Line Engineering. To obtain maximum benefits from the SPLE development methodology, businesses need to implement coordinated changes in development methodologies, tools, product architectures, organizational designs, and business models. The SPLE Reference Model involves higher levels of abstraction than the engineering of single systems partly because the platforms require substantial investments, have long life cycles, and have to provide product line architectures and features generally applicable to a wide range of products, services, and markets. Without appropriate abstractions, the platforms with predefined variability cannot be built and managed effectively. On the other hand, standardized methods and tools for developing product lines cannot be easily adopted, tool vendors face difficulties in developing tools to enable SPLE, and universities cannot effectively set up SPLE courses because an internationally accepted curriculum is missing. However, two surveys and workshops are actually organized to derive recommendations for educators to continue improving the state of practice of teaching SPLs, aimed at both individual educators as well as the wider community [17]. The International Organization for Standardization (ISO) has initiated several projects to create a set of international standards for SPLE, such as the ISO/IEC 26550 Reference Model, where guidelines and available tools and techniques are provided. ISO hopes that researchers and practitioners can enrich this initiative.

Our research aims to make a practical use of the SPLE methodology, integrating practices such as the early consideration of quality issues in the RA design, and applying it to an industrial case study.

Guidelines, practices, benefits, and risks to adopt a SPL approach are discussed in the SEI SPL framework by Northup and Clements [6] "… substantial production economies can be achieved when the systems in a SPL are developed from a common set of assets in a prescribed way, in contrast to being developed separately, from scratch … It is exactly these production economies that make the SPL approach attractive". Reuse has evolved from subroutines (1990s) to services (2000s). Most organizations produce families of similar systems, differentiated by features, and a reuse strategy makes sense. Commonalities are exploited differently according to the enterprise domain; hence the study of the domain is essential for the SPL development. Enterprises that have succeeded with product lines vary widely in considering activities such as the nature of their products, the market or mission, business goals, organizational structure, culture and policies, the software process discipline, and the maturity and extent of legacy artefacts. Nevertheless, there is not one correct set of practices for every organization [4]; activities and practices emerge, having to do with the ability to construct new products from a set of common assets while working under the constraints of organizational contexts, and this problem is still not solved. The recommendation in this work to derive a concrete product, is to follow a production plan, which details how the core assets are to be used to build the product. However, each of the main activities for SPL development, saying, the core asset development (where the RA is built), the product development, and management is individually essential, and they are a blend of technology and business practices. However, the threshold between these activities and how to put them smoothly and correctly together is not discussed in the framework, and it is the problem we are focusing. We are putting into industrial practice an SPL development approach that blends several practices and techniques: a bottom-up architecture-cantered strategy to build the RA [7], the extraction of functional components from business goals, the use of the properties of the domain architectural styles, the early account of quality requirements as architectural components, as the major responsible of the SPL variation. In our work we use the Domain Realization guidelines of [3], [4], studying the available technology and developing the component interfaces; our goal is to reduce the gap between the abstract RA and the concrete product derivation, to produce in the near future, a semi-automatic product configuration process.

The paper of Dai, He and Xing [15] proposes a "6+1" structure as a cloud service platform for Human Resource Management; it does not concern SPL, only the HR domain. HR management includes standard management tasks such as wage, attendance (including leave and vacation demands), and personnel file. A rather classic 6-layered structure is proposed for the platform architecture: 1. UI layer to face clients and to collect data; clients choose the services (e.g. salary, vacation request or recruitment management). 2. Data integrity control layer. 3. Process layer for data processing; it is the platform computation core. 4. Data base logic layer, to control the operations of data in data tables. 5. Data table definition layer, and 6. Data Base layer. Our interest is focused on the "+1" layer. In practice work, HR management business is not always normalized, such as organizational structure adjustment, salary system design and psychological counselling to the employees; the laws are not normalized neither and can be applied differently to each entity. These problems usually require relevant experts to solve, based on their experience, implying huge costs. Meanwhile, many experts have characteristic methods and techniques, they need working platforms to provide services. The cloud service platform can help them build their own sub-working platforms and offers technical support. The "+1" structure will implement characteristic HR management service. The *Configuration* component in our *Context* external system for the Vacation Request SEDIT subsystem, could fit into this "+1" layer since it will be used by external experts, usually located in third-party enterprises.

The adoption of the cloud technology for HR systems is the subject of Kumar work [16], but it does not concern SPL. The National Institute of Standards and Technology (NIST) defined cloud computing as "a model that provides ubiquitous, convenient, on-demand network access to a shared pool of computing resources like servers, networks, storage, applications and services with minimal management effort and service provider interaction". The most updated version of the cloud technology in the field of HR is the Software as a Service (SaaS) technology. This is the most popular form of the technology. The server, in this case, usually provides the entire software to the user through an application, which does not need to be installed or upgraded because the vendor automatically does this onto the cloud. The user has only to upload and manage information stored in the cloud. Neither upgrading nor updating need to be done by the user. In our case, the whole HR SEDIT system of Berger-Levrault seeks to be offered though the cloud as a SPL in the HR domain. The Vacation Request subsystem is an important functionality that is common to many entities covered by SEDIT. A huge configuration effort must be performed to offer the appropriate configuration to the great number of SEDIT clients. A SaaS configuration service should be included, starting to configure the Vacation Request subsystem to face the law's evolution, being this a major problem of HR systems. Other configuration problems could be solved in this way; but this subject is still an on-going work.

IV. Conclusion

In a previous work [7] we have designed a RA for a HR SPL. We have now proceeded to link the high-level RA core asset to the lower level application design by adapting the Domain Realization phase of SPLE [3], [4]. An industrial case study, the Vacation Request subsystem of the SEDIT system of Berger-Levrault illustrates our approach.

The main result of this preliminary work is a new version of the RA including the specification of the component interfaces. In particular, we studied how the non-functional abstract components that were introduced to take in charge qualities, are realized using technological solutions. A problem that still remains, is to define an external configuration component that will be used to satisfy laws evolution, which is a huge problem in HR systems. The perspectives would be to have the SEDIT HR system on the cloud and an external system to take account of law changes using Java injection and design patterns facilities, without modifying the code [18].

References

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

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

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

[4] ISO/IEC NP 26550: Software and Systems Engineering – Reference Model for Software and Systems PL. ISO/IEC JTC1/SC7 WG4, 2015.

[5] R. Mazo, S. Assar, C. Salinesi and N.B. Hassen, "Using SPL to improve ERP Engineering: Literature Review and Analysis", In Latin American Journal of Computing LAJC, Vol. 1 (1), 2014.

[6] L. Northrop and P. Clements with 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, Carnegie Mellon, 2012.

[7] M. Derras, L. Deruelle, J. M. Douin, N. Levy, F. Losavio, Y. Pollet and V. Reiner, "Reference Architecture Design: a practical approach," ICSOFT 2018, Porto, Portugal, pp. 599-606, July 2018.

[8] Berger-Levrault, Schéma architecture - SEDIT REST – Evolution, Internal Communication, 2017.

[9] ISO/IEC 25010: SQuaRE, Quality model, 2011.

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

[11] E. Gamma, R. Helm, R. Johnson, J. Vlissides and G. Booch, Design Patterns: Elements of Reusable OO Software 1st Edition, 1995.

[12] G. Rodríguez, Analyse fonctionnelle de domaine dans le cadre de logiciels de gestion de collectivités territoriales, Master International, École d'Ingénieurs, CNAM (Conservatoire National des Arts et Métiers) Paris, France, September 2017.

[13] D. Bell, UML basics – The Component Diagram – UML 2.0 diagrams, IBM, 2004. https://www.ibm.com/developerworks/rational/library/dec04/bell/index.html

[14] T. Käkölä, "Standards initiatives for SPL engineering and management within the international organization for standardization," System Sciences (HICSS), 43rd Hawaii Internat. Conf., IEEE pp.1-10, 2010..

[15] L. Dai, Y. He, G. Xing, "Intelligent Information Management," 7, pp.1-6, published in SciRes Online January 2015. http://www.scirp.org/journal/iim, http://dx.doi.org/10.4236/iim.2015.71001.

[16] R. Kumar, "Cloud Technology and HR Management," Annual Research Journal of SCMS (Symbiosis Centre for Management Studies), Symbiosis International University, Pune, India, Vol. 5, pp. 82-91, March 2017.

[17] M. Acher, R. E. Lopez-Herrejon, and R. Rabiser. "Teaching Software Product Lines: A Snapshot of Current Practices and Challenges,". ACM Trans. Comput. Educ. 18, 1, Article 2, 31 pages. October 2017, https://doi.org/10.1145/3088440

[18] Douin J. M., Pollet Y.. La proposition VIP : Variability & Injection Pattern, Internal communication, Cédric, CNAM, 2018.