Modeling a distributed environment for a petroleum reservoir engineering application with software product line

Rafael de Faria Scheidt, Patrícia Vilain, M. A. R. Dantas
Research Laboratory of Distributed Systems (LaPeSD)
Department of Informatics and Statistic (INE)
Federal University of Federal Santa Catarina (UFSC)
Florianópolis, SC - Brazil

E-mail: {rfscheidt, vilain, dantas}@inf.ufsc.br

Abstract. Petroleum reservoir engineering is a complex and interesting field that requires large amount of computational facilities to achieve successful results. Usually, software environments for this field are developed without taking care out of possible interactions and extensibilities required by reservoir engineers. In this paper, we present a research work which it is characterized by the design and implementation based on a software product line model for a real distributed reservoir engineering environment. Experimental results indicate successfully the utilization of this approach for the design of distributed software architecture. In addition, all components from the proposal provided greater visibility of the organization and processes for the reservoir engineers.

1. Introduction

The reuse of elements and components inside projects with a similar domain scope in software engineering and object oriented programming approaches is a common practice. This effort is related to the reduce of costs, time and productivity [1]. Aiming to reach these goals several techniques were proposed, such as: domain engineering, frameworks, patterns, software architecture and also development based on components [2].

The software product line (SPL) is a characteristic approach which was conceived to organize systematically and also provide a predictable fashion to software reuse based on the same product domain. As mentioned in [3], the SPL technique appears from products or similar researches in the same domain to a common business area. In accordance to the Software Engineering Institute (SEI) [1], the SPL paradigm represents software systems which share a set of common and controlled characteristics. These satisfies a special market segment and are developed from key artifacts (core assets), in a pre-defined form.

Several organizations, when implementing their software domain applications, adopt some particular characteristics to differentiate one specific product [18]. Thus, these organizations are always seeking for processes and approaches that could help in the reuse of the main characteristics from a specific product.

This paper presents a research work which is depicted by a design and an implementation of software product, for a reservoir engineering environment, adopting the software product line (SPL) paradigm.

The contribution is characterized for considering a real application, which was being developed at the Research Laboratory of Distributed Systems (LaPeSD) at Federal University of Santa Catarina. In this group, with large experience in developing distributed applications, similar to other peers groups, when a new version of a product was demanded several artifacts were reused, albeit observing an evolution of some characteristics. Therefore, it is possible to verify the existence of core architecture with some components variants.

In addition, it was frequently the case where the client asks for a specific feature and afterwards asks to remove this facility. After sometime, the client claimed again for that specific module, but now the system could not support it. In other words, in both sides, clients and developers, there were no culture of reuse. Therefore, the utilization of the SPL approach may improve operations such as inclusion and remove of elements from the distributed architecture. As a result, it was expected to achieve a better level of flexibility, maintenance and product evolution.

The paper is organized as follows. Related works are presented in section 2. In section 3, we show the conventional distributed configuration utilized for the development that is contrasted with the proposal considering the use of the SPL technique. Empirical results are presented in section 4. Finally, in section 5 we presented conclusions and future work related to this research.

2. Related Work

In this section, it is presented two research projects in which distributed configurations were conceived as an intrinsic feature together with the SPL approach. These two related works were chosen because it was possible to have more access to information from these proposals.

2.1. UBÁ

The UBÁ proposal [14] aims to build SPL architecture to help the development of middleware systems to the grid computing configurations. The approach is used to minimize the complexity related to this configuration. Examples of some facilities created are: the coordination for large scale resource sharing; Quality of Service (QoS) provision; support of a large number of heterogeneous devices; geographic dispersion of resources, people and applications.

This research was characterized through the SPL instantiation of the proposal, thus creating a middleware for grid computing which provides their main facilities. In addition, it was built a structure that utilizes the API from the created middleware. The configuration was tested in different scenarios, executing successfully one application in these different environments.

2.2. Systemized Reuse of SPL in Financial Segment

A study presented in [16] covers a system reuse, through the utilization of the SPL, in the financial segment. This document illustrates studies referring to the five major Brazilian banks. The key element of this research is to present how the reuse is adopted inside financial sector and how this practice contributes to the success of software projects.

An interesting example that pointed out the necessary use of reuse approach is illustrated by a specific bank that executes one modification in ordinaries accounts and it was required modification into 72 systems. This problem was identified as a lack of planning in the reuse inside those systems.

2.3. Observed Aspects

The UBÁ approach focuses its development for a specific application domain as a grid computing middleware. In other words, domain engineering for a broadly distributed system is not covered.

On the other hand, the research project presented in [16] illustrates that the SPL approach is not a key element for those organizations. The functions of coupling and maintenance for the applications do not consider a software product line, and also it exist a lack of reuse as macro paradigm.

Understanding the former two contributions, which could have some intersection aspects with the target configuration of our research, it was essential to conceive a differentiated contribution considering a more generic distributed system environment.
Therefore, an effort in which can be possible represent the domain engineering and application instantiation for distributed system configurations, may answer our research question. As result, our proposal should conceive to design and implement a software environment approach based on the SPL paradigm.

It was not found in the literature a similar proposal, considering a real case study, in which a generic distributed system had a support from the SPL with a high level of artifacts reuse.

3. Proposal

In this section, it is described the research proposal in terms of design and implementation of the software product line approach for a real distributed system configuration. Thus, the section shows how was the old development process for configuration, the validation for the job management system and how was the reformulation from the old-fashioned to the new paradigm.

3.1. The Original Development Environment

The development of an application for any large distributed system configuration can be seen as a complex process. As mentioned in [17], heterogeneity in terms of hardware, operating systems and resource management systems (RMS) are three examples of key aspects to be considered for any distributed project. As a result, elements such as graphic interfaces, network interfaces and also programming middleware software packages are important to be analyzed.

The main contribution of this research work is to apply the SPL Smarty process in a real distributed system project, which is being developed at the Research Distributed System Laboratory (LaPeSD), at the Federal University of Santa Catarina (UFSC), in collaboration with a petroleum engineering company.

As a result, the proposal is to contribute for the systematization of future developments considering artifacts, components and architecture reuse.

In this context, each element (e.g. GUI, interfaces and RMS) could be translated to components forming a common architecture known as domain engineering (DE). Afterwards, for any new product will be possible to have an instance analysis for this new application engineering (AE) architecture, designing a specialization for those no common parts. Through the utilization of the SPL approach, it is possible to have a systematic process, adopting the reuse paradigm and also a planning procedure to add these new components to the architecture. Important to observe, these features will be added into the configuration without an extra effort.

However, to reach level of abstractions mentioned before, it was required changes in the actual architecture. The original development environment was characterized as a tightly coupled configuration and it was difficult apply the reuse and componentization techniques. Figure 1 shows a macro idea, on how was the configuration.

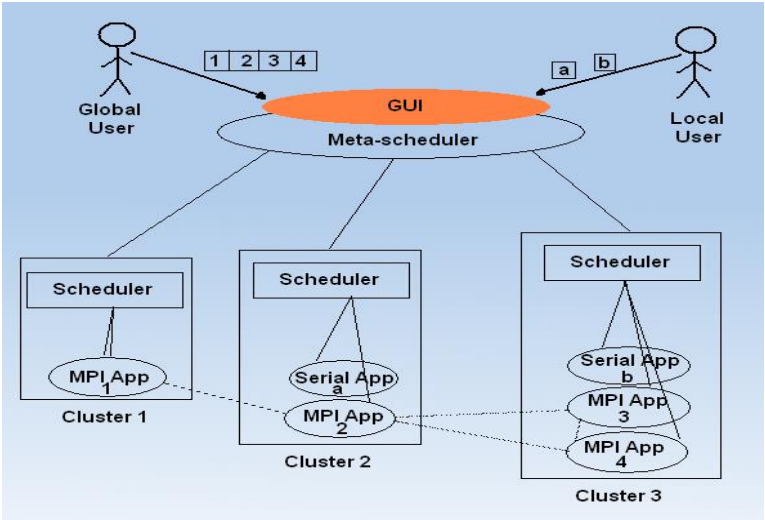

Figure 1. Macro view of the original development environment.

A meta-scheduler element has a key importance in the distributed configuration. This component provides the interface between requirements from a user to a scheduler (RMS), in a specific cluster configuration. Usually, the meta-scheduler is coined as the middleware of the configuration. The main reason for this is based on the fact that each cluster has its own scheduler (e.g. Condor, OpenPBS and LSF). Thus, the meta-scheduler is obliged to know each API from all schedulers and afterwards realize all communications.

In a case of a new cluster added to the configuration, the middleware is informed and all elements necessary to this communication should be provided by the middleware. Location, ports, components and general descriptions are examples of elements necessary for all communications. Therefore, this is one new component that should be aggregated to the middleware environment, and which also should be available to interoperate with other existing components.

This contribution can be considered differentiated, because it proposes an approach which can be translated in a systematic method, reusable and with previous planning using the SPL paradigm. As a result, it will be possible to add components to the original distributed system architecture without large changes for the middleware environment.

To reach the necessary transparency, it is necessary to process some changes into the original middleware architecture. In other words, our contribution presents a new architecture approach with a reusable function and components with variations points that were previously defined. These definitions were designed foreseeing new functionalities, components and schedulers.

3.2. The New Development Environment

The ordinary process of software product line is shown in figure 2. In each phase of the process, artifacts are produced and stored inside a SPL repository.

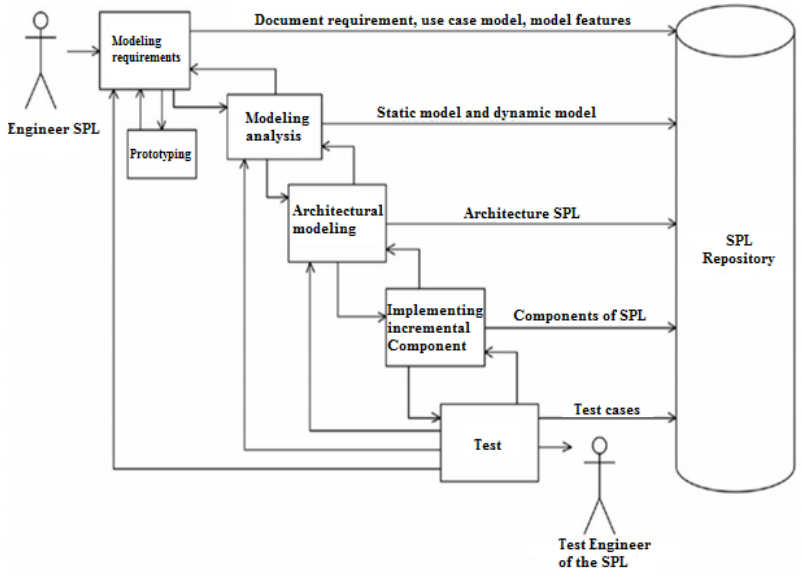

Figure 2. SPL developing process.

The process starts with the requisite modeling technique which comprises the model elaboration and requisite document, use cases models and a feature model. Use case models define SPL functions requisites in terms of actors and use cases. It is also possible to realize the similarity and variability model, in this research study, from the Smarty approach. Inside the features model, each item is a requisite that defines one similarity or variability which could be translated into parameters.

In the requisite analysis phase, all SPL functional and non-functional requisites are specified. These requisites are verified in the next steps to check their relations. Thus, it is established restrictions through pre-requisites. Found all pre-requisites, these will guide the direction of the development, for build the architecture and its validation.

3.2.1. Functional Requisites

1. RF001 – Advanced Resource Reservation.

Description: The system should be able to provide an advanced resource reservation function for a task to be executed in the future with the required resources.

2. RF002 – Immediate Resource Reservation.

Description: The system should be able to provide immediate resource reservation function

3. RF003 – Resource Reservation Based on Characteristics.

3.2.2. Non-Functional Requisites

4. RNF001 – Add/Remove enhancements or functionalities with impact the actual structure.

Description: The structure should be able to support the add and remove functions to improve existing functionalities of the architectures. It is also possible to add functionalities and components to the environment without any prejudice for users and services. Example of such prejudice is the communication between services.

5. RNF002 – Heterogeneous Services.

The architecture should support any operating system and hardware

Departing from functional and non-functional requisites it is possible to identify use cases that exist in the domain model. Before the execution of the use cases, it is imperative to identify all possible actors in the system as illustrated in figure 3.

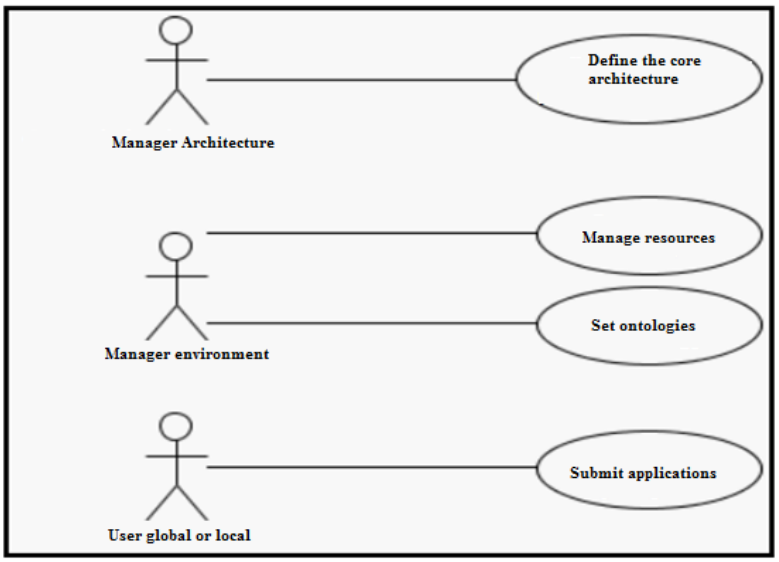

Figure 3. Actors from the use case model.

All use cases require one refined action and specific actions from each actor, which should be explained in their user cases.

The use case evolved in figure 4 presents innumerous variance points such as: “Define the main architecture”; “Define GUI”; “Define meta-scheduler”; “Define LaPeSD products”; “Define RMS”. A special fact to be considered is the occurrence of obliged activities as “Define rules” and optional activities such as “Allows dynamic changes”.

In addition to the variation point “Define meta-scheduler”, there are two variants with stereotypes “alternative XOR”. This indicates that only one option is possible to be chosen. The specification exists as exclusive alternative relation inside the SMarty approach.

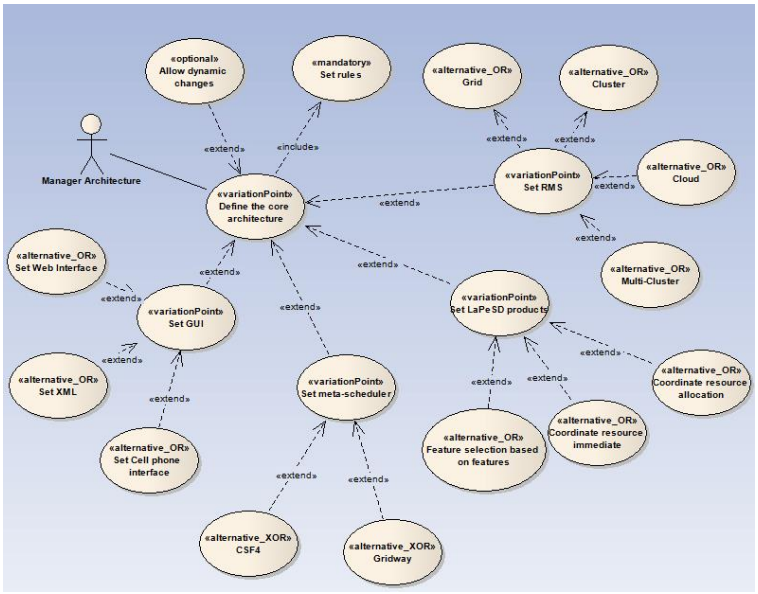

Figure 4. Variability.

The architectural modeling is an important phase in the SPL approach. This phase is characterized by a specification and definition of the architecture based on components. In this case, the analysis model is addressed to the architecture model taking into consideration software patterns.

Figure 5 shows the proposed architecture after applying the SPL approach for the distributed system environment. The vision layer is composed by the communication and job submission interfaces. In addition, a user may select resources and necessary attributes. After this layer, it is presented the main variation point from the architecture. This communication interface adopts as variation points some general Communicators, such as Condor, SGE and LSF. These items if instantiated in a domain engineering, it will be implemented as their RMS specifications. This new feature allows that new RMS may be added to the architecture. The implementation of these interfaces is a product from the LaPeSD.

4. Experimental Results

In this section it is presented some experimental results, after adopting the SPL approach in the project. The evaluation of proposal was realized through study case in the real project from the laboratory. Therefore, all information related to the architecture, components and artifacts are real.

The complexity of the environment was bypassed, first conceiving a model for the graphical user interface in which was possible to submit jobs. As a result, our effort was to add a special focus in the development, implementation and validation of this layer, due to its key importance.

The access to the environment was conceived through the use case from the interface layer. It was utilized variation points and delimitated variances using the SPL. The SmartyProcess approach was the choice which is represented by: web interface, cellular interface and WebService. These elements will be the family products from the LaPeSD, considering the layer interface with the user.

In addition to previous interfaces, it was designed a communication interface abstraction with several RMS that exist and could be possible for the architecture. Condor, SGE and LSF communicator and variance were implemented. Each communicator specifies the communication form inside each RMS, where each one has its commands.

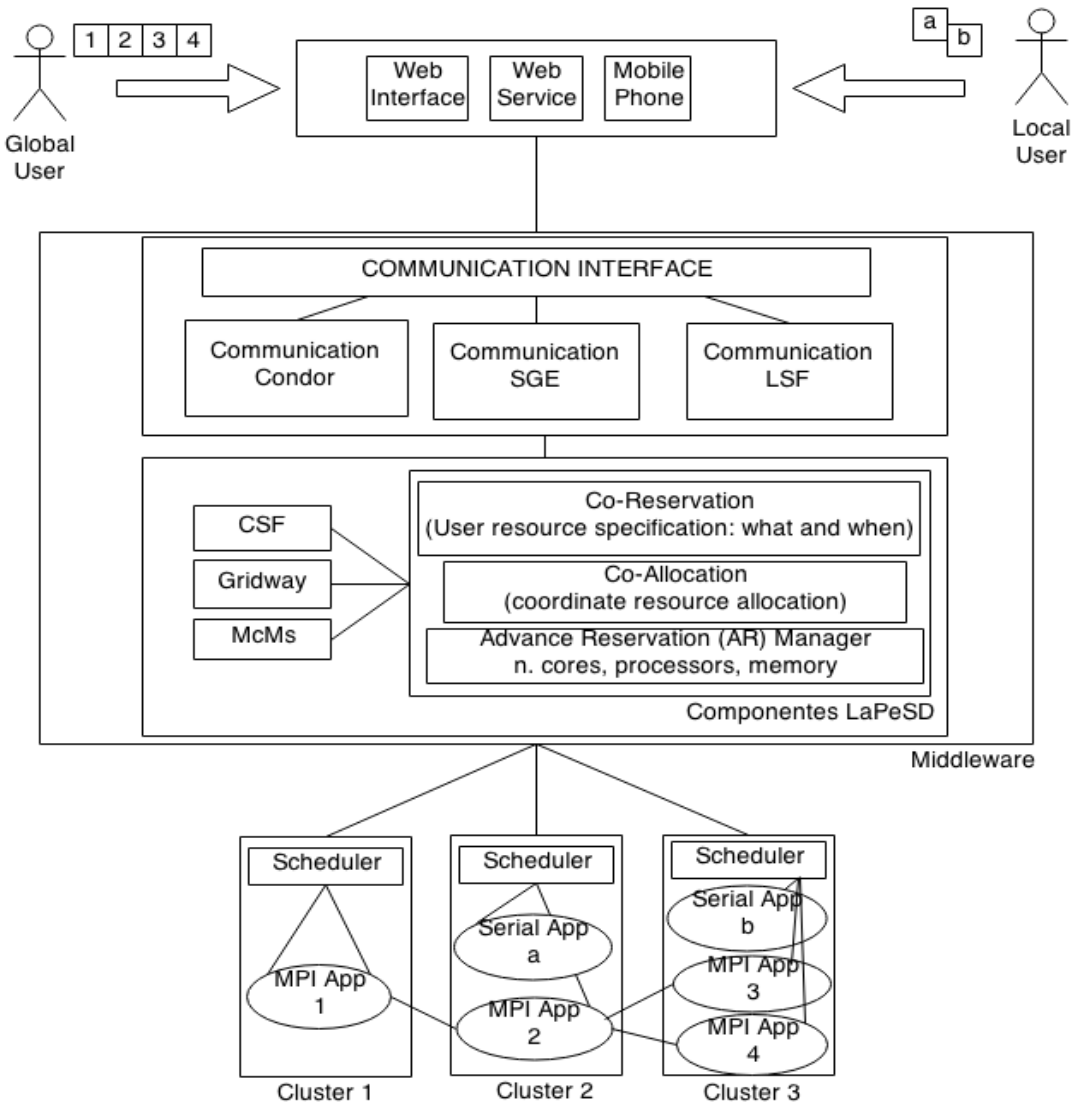

Figure 5. Proposed architecture.

The communication procedure was decoupled from the interface and created one structured in edge of the middleware. As a result, components from the interface were turned into components and were open to new interface designs or evolutions. An example, it could be seen in the future a new graphical interface considering dynamic resource management adopting a fuzzy logic paradigm to hardware and software selection.

A pool was realized among experts from the laboratory and it was possible to map messages that are usually displayed to end users. These were implemented in the communication interface for the graphical interfaces for each communicator as shown in table I.

It this phase of our work it was created a webservice, which made possible any implementation in the upper layer (or GUI). Therefore, it was possible to utilize all messages from table I, where a Global or a Local User was able to submit jobs, process some management tasks or a RMS.

Table 1. EXISTING MESSAGES IN THE RMS COMMUNICATION.

Message	Function	SGE Command	Condor Command
submitJobSynch()	This method is called for a syn submission	Qsub	Condor_submit
delJob()	Use to cancel/delete jobs	Qdel	Condor_rm
getJobs()	List job in execution mode	Qstat	Condor_q
listNodes()	List information status about nodes and execution nodes	Qhost	Condor_status

Afterwards, the communication interface which is represented by the Communication class inside the webservice, receive a request this is forwarded to the CommunicationServiceImpl class. This latter will verify which RMS will be manipulated. The submission through the user GUI requires an indication of a RMS. As soon as a requests arrives at the webservice, following the request it knows which class will be invoked, i.e. CondorService or SGEService. In case that the class CondorService was chosen, this component will be communicating to the Local RMS and will be processed their own specifications and requirements. As an example, if a job submission is required the specific command of the Condor RMS will be executed, without the necessary knowledge of any syntax from the end user.

5. Conclusions and Future Work

In this research work it was presented a design and implementation of the software project line approach in a real distributed system project. The goal was to provide in a systematic fashion components, also to create domain engineering. The aim was successfully reached.

Before the present effort, the original development environment was characterized as a tightly coupled configuration and it was difficult apply the reuse and componentization techniques. One frequently occurrence was that client asks for a specific feature and afterwards asks to remove this facility. After sometime, the client claimed again for that specific module, but now the system could not support it. In other words, in both sides, clients and developers, there were no culture of reuse. Therefore, the utilization of the SPL approach has improved operations such as inclusion and remove of elements from the distributed architecture. This objective was reached through a layer, where all requests were pre-processed and redirected to an appropriate RMS module. As a final result, the conceived SLP environment provides an enhancement artifact reuse approach where errors were lower and software conformity was higher. In addition, characteristics such as an easier maintenance and less time to develop features were verified in the new configuration.

As a future research plan, we are going to implement the SPL approach in ubiquitous part of the software environment. In addition, we are going to develop a set of tools to provide some support and automatization for the design of applications. It will be also important conceive and implement a tool which may provide an enhanced option to select components and variants of distributed systems, thus improving applications creation.

6. References

[1] SEI, Software Engineer Institute [online]. Available: http://www.sei.cmu.edu/, 2013
[2] Gimenes, I. M. S.; Travassos, Guilherme Horta. The Product Line Approach to Software Development. Ingrid Jansch Porto. Porto Alegre, Brazil, 2002.

[3] Vasconcelos, A. One approach to support the creation of reference architectures based on domain analysis of Legacy Systems. Phd Thesis, Federal University Federal of Rio de Janeiro, Rio de Janeiro, Brazil, 2007.
[4] Dantas, M. High Performance Distributed Computing: networks, clusters and computational grids.Rio de Janeiro: Axcel Books, Brazil, 2005.
[5] De Rose, C. A. F.; Ferreto, T. C., Improving Performance Analysis Using Resource Management Information, Springer Berlin/Heidelberg, 2003.
[6] Buyya, R. Grid Computing Info Centre (GRID Infoware). Available: http://gridcomputing.com, 2011.
[7] Colvero, T. A.; Dantas, M.; Cunha, D. P. da. Environments clusters and computational grids: features, facilities and challenges. Criciúma: Unesc, Santa catarina, Brazil, 2005.
[8] Yeo, C. S. et al. Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers. 2006
[9] Clements, P.; Northrop, L. Software Product Lines: Practices and Patterns. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.
[10] Northrop, L. M. SEIs Software Product Line Tents. IEEE Software, v. 19, n. 14, p. 32-41, 2002.
[11] Pohl, K.; Bockle, G.; Linden, F. J. V. D. Software Product Line Engineering: Fundations, Principles and Techniques. Secaucus, NJ, USA: Springer-Verlag, 2005.
[12] Oliveira Junior, E. A.; Gimenes, Itana M. S.; Maldonado, José C. Systematic Management of Variability in UML-based Software Product Lines. Journal of Universal Computer Science, p. 1-20, 2010.
[13] Oliveira, A. Edson SystEM-PLA: A Method for Evaluating Product Line Architecture Software Based on UML, Instituto de Ciências Matemáticas e Computação, State University of São Paulo, Brazil, Ph.D. Thesis, 2011.
[14] Oliveira, D.; Rosa, N. Ubá: A Software Product Line Architecture for Grid-Oriented Middleware. Informatic Center, Fed. Univ. of Para, Santarem, Brazil.33rd Annual IEEE International Computer Software and Applications Conference, 2009.
[15] Bosch, J. Preface. In: The 2nd Groningen Workshop on Software Variability Management: software product families and populations, 2004, Groningen, The Netherlands. Proceedings. Groningen, The Netherlands, 2004.
[16] Reinehr, Sheila dos Santos. Systematic reuse of software and Software Product Lines in the financial sector: a case study in Brazil.2008. 327 f. Escola Politécnica, Programa de Pós-Graduação em Engenharia de Produção, Phd Thesis, São Paulo, Brazil,2008.
[17] Silva A P C and Dantas M A R 19th International Symposium on Computer Architecture and High Performance Computing, (SBAC-PAD'07) 143-150, 2007.
[18] Sugumaran, V.; Park, S.; Kang, K. C. Software Product Line Engineering. Communications of the ACM, v. 49, n. 12, 2006.