2009 Fourth International Conference on Software Engineering Advances

An Ontology-based Approach to Model-Driven Software Product Lines

Nuno Ferreira
Dep. Sistemas de Informação
Universidade do Minho
nuno.ferreira@dsi.uminho.pt

Ricardo J. Machado
Dep. Sistemas de Informação
Universidade do Minho
rmac@dsi.uminho.pt

Dragan Gašević
Athabasca University
dgasevic@acm.org

Software development in highly variable domains constrained by tight regulations and with many business concepts involved results in hard to deliver and maintain applications, due to the complexity of dealing with the large number of concepts provided by the different parties and system involved in the process. One way to tackle these problems is thru combining software product lines and model-driven software development supported by ontologies. Software product lines and model-driven approaches would promote reuse on the software artifacts and, if supported by an ontological layer, those artifacts would be domain-validated. We intend to create a new conceptual framework for software development with domain validated models in highly variable domains. To define such a framework we will propose a model that relates several dimensions and areas of software development thru time and abstraction levels. This model would guarantee to the software house traceability of components, domain validated artifacts, easy to maintain and reusable components, due to the relations and mappings we propose to establish in the conceptual framework, between the software artifacts and the ontology.

Software Engineering Process; Software Design

I. INTRODUCTION

Highly variable business domains (like insurance) faces many and specific problems, that derives from the weight of legacy technologies in the information system, thru the way the contact with the outside world is made, passing by domain-specific variability. Applications are traditionally built in a straightforward way, many times having no relation whatsoever between them. When dealing with legacy systems that problem emerges even more, due to the time elapsed between the requirements specification for each developed module and the present. Usually, each application has its own configuration, store its own data and is guided by its own business rules [1].

One of the major costs in software is its maintenance, as it becomes more complex and expensive thru time. In highly dynamic domains, where a great maintenance effort is needed to keep the software up-to-date, the overall cost rises and the time-to-market is critical, there is a need to improve the development process and manage the variability and reusability. One of the key challenges is to understand and identify the relations and representations of software artifacts and resources involved in the maintenance [2], even tough all the efforts done to create a well established and defined development process, the traceability is often lost. This is mainly due to the fact that the software artifacts involved in the development process are often written using different programming or modeling languages and the process nor the tools involved support traceability links between artifacts and development phases [2].

To address the previously exposed problems, the current proposal is to deliver a conceptual framework to the software development based on models in a product line scenario supported by ontologies. Combining software product lines with a model-driven approach, a highly-customizable, reusable and generative software development environment can be achieved. The desired solution is the integration of ontologies for the purpose of more effective model-driven development in insurance product lines.

In software engineering, the conceptual model underlying the different business domains (like banking, insurance, industry, and others) need to be explicitly defined by ontologies. Why use ontologies? An ontology provides a vocabulary [3], for referring to the terms in a given domain or subject area. When dealing with a dynamic domain that does not have a fixed and agreed vocabulary, trying to define one without the inherent control of ontologies would generate ambiguity, lack of consistency and absence of hierarchy. Having reusability in consideration, ontologically-defined knowledge of a domain can be shared and reused.

The chosen research approach will use Action Research [4] and Experimental Software Engineering [5] to evaluate results. Action research is used in an enterprise environment and the outputs can be adapted to other domains.

II. STATE-OF-THE-ART

The state of art of this work relates to essentially three main areas: Model-Driven Development, Ontology and Software Product Lines.

A. Model-Driven Development

Model-Driven Development (MDD), a common name for Model-Driven Software Development (MDSD) [6], aims at the provision of practically applied modules for the software development processes, regardless of tools, in model-driven approaches [6, 7]. Model-driven development is generic to the development of everything, based on models, while model-driven software development is specific to software realities. In this document we will always refer to model-driven in the context of software.

The software engineering discipline that deals with this subject is called Model-Driven Engineering (MDE), in which the process strongly depends on using models, as can be seen in [8, 9]. One OMG initiative called Model Driven Architecture (MDA) [10], is considered a possible metamodeling architecture that enables the use of MDE

978-0-7695-3777-1/09 $26.00 © 2009 IEEE
DOI 10.1109/ICSEA.2009.88

principles. MDE centers on models. A model is interpreted as a description or specification of a certain system for a specific purpose [11], a set of statements about some system under study [12]. A modeling language (e.g. UML, EMF) is used to specify models and they can be defined by metamodels. A metamodel is a model of a modeling language. It makes statements and constrains what can be expressed in the valid models of a given modeling language [12]. The relations between models and metamodels can be expressed thru metamodeling architectures. These usually are composed by three layers, like in MDA. The layers are:

- M1 layer, also called model layer, it's where models are defined by using modeling languages;
- M2 layer also called metamodel, where the metamodels, or models of the modeling languages, are defined. Examples of models of modeling languages are UML and EMF.
- M3 layer also called metametamodel layer where the only metamodeling language is defined. Examples of metamodeling languages are MOF (for UML) and Ecore (for EMF). We say the only metamodeling language in this layer in order to give a unique grammar space for defining various modeling languages on the M2 layer.

The relation between the layers is of the instance-of or conformant-to type. This is to say that a model is an instance-of (or conformant-to) a metamodel and a metamodel is an instance-of (or conformant-to) a metametamodel. By using only one metamodeling language, the various modeling languages that are instances of if, share the same structures and can be processed the same way, for instance, using the same interface.

One important aspect related to MDA is model transformations [13]. Transformations are categorized as Model-to-Model (M2M) and Model-to-Code (M2C) [13]. M2M transformations translate between a source and a target model. M2C transformations can be seen as a special case of M2M transformations with extra information about the target programming language and translate a model to text. OMG's defined a standard for M2M transformations, named Query/View/Transformations (QVT) [14].

For some authors, model-driven software development is considered "the first true generational shift in programming technology since the introduction of compilers" [15], able of really changing the way of developing applications [16].

B. Ontology

The term "ontology" originates from philosophy, where it denotes the study of existence [15]. In computer science, the most common definition has been provided by Gruber [17], which says that "An ontology is an explicit specification of a conceptualization". Corcho [18], extended that definition, by saying that an ontology is "a formal, explicit specification of a shared conceptualization". Conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use, are explicitly defined. Formal refers to the fact that the ontology should be machine-readable, that is to say that a machine can process it. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group. It is the consensus of a community of experts. Also Gasevic [3] reinforces that ontologies can be interconnected by themselves, opening a perspective on ontological mapping.

Ontologies are used to model static knowledge [18] while problem solving methods specify generic reasoning mechanisms. A common case is the relationship between a user account in a given information system and the corresponding real person. It is not easy to determine how many roles this person has in the system. He can play a role of a customer, the person to be compensated by an accident and so on. The person can have multiple accounts in the different systems, each according to a given role. Modeling users and their roles in the different systems requires an extensive ontological modeling work. In complement to this reasoning, ontologies can help to configure new systems from existing reusable components.

According to Uschold [19], an ontology may take a variety of forms, but necessarily it will include a vocabulary of terms, and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretations of terms. There has been an increased interest in ontologies as artifacts [20] to represent human knowledge and to work in critical sections of applications.

According to Sure and colleagues [21], the three main ideas of ontology content evaluation are: Ontology content should be evaluated for it's the entire life cycle; The content evaluation should be support by ontology development tools in the entire ontology building process and; There is a strong relation between content evaluation and the underlying knowledge representation paradigm of the language in which the ontology is implemented.

Gasevic [3] points out that ontologies provide a number of useful features, like:

- Vocabulary: a controlled vocabulary, referring to terms in a business area;
- Taxonomy: hierarchical categorization or classification of entities within a domain;
- Content Theory: ontologies go further than the sole identification of classes, relations and associations. They provide that information in an elaborate way, using ontology specification languages;
- Knowledge Sharing and Reuse: besides providing the description and relation of concepts in a domain, ontologies also promote their sharing among agents and applications.

Software engineering ontology defines [20] common sharable software engineering knowledge including particular project information. It defines concepts, and the way they are related or can be related. It can be possible to achieve a common and consensual vocabulary between members of a project. With ontologies, it is possible to represent and communicate software engineering knowledge and project information.

Ontologies and models are related in the software development process, since a metamodel is [3] an explicit model of the constructs and rules needed to build specific models within a domain of interest. This can be mapped to an ontology, by relating constructs and rules with entities and relationships.

C. *Software Product Lines*

Along the years companies have placed a great deal of effort to produce software artifacts that can be reused in other projects [22-24]. In relation to software product lines (SPL), the effort to develop a single product is sometimes similar to the one needed to develop a line of products [25] able to produce several artifacts. They need to reorganize their development strategies to promote reuse and mass-customization [24]. The difference between two or more products is so minimal that it is financially sustained to develop a line of products instead of a single one. In the insurance domain many products are almost identical, differentiating only in some particularities.

Traditional software development is usually performed one system at a time [25], being focused on delivery, and not caring much on its evolution. This leads to several problems: failing to deliver the project on time and on budget; its quality is sometimes arguable; the maintenance cost is high; and all this leads to a decrease of competitiveness. When considering the previously exposed, some objectives could be defined [25], without any order:

- Reduce the development cost: if the software cost doesn't decreases, it may no longer continue to be pervasive as now;
- Increase quality of software: increasing reliability, maintainability, efficiency and other overall system quality attributes must be considered;
- Decrease time-to-market: if the time it takes to deliver a product on the market is reduced, there are more profit opportunities;
- Decrease maintenance cost: it is needed a fundamental change in development to reduce maintenance cost of installed products.

One possible solution to achieve these objectives is thru reuse of software. Reuse should be [25] opportunistic, carefully planned and proactively thought. It should also employ a top-down approach, that is, it must follow a master building plan, not discarding cases when bottom-up decisions must be made – the case when a implementation decision affects the architecture of the system.

According to some authors [23, 26], SPLs have been probably the software development paradigm shift that brought the biggest revolution since the advent of high-level programming languages. A SPL is [26] a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that dare developed from a common set of core assets in a prescribed way. These last words, "prescribed way", are what distinguish the previous product line definition from the traditional one. They points out the need to follow guidelines for developing assets, not let them be built in an arbitrary fashion, separately, from scratch [26].

A product line focus on promoting strategic high-level reuse, which is planned, opportunistically placed, not fortuity. Product lines are based on more than components. They involve planning, management, and more. In a product line there are multiple simultaneous products, each with its own versioning and lifecycle. Previous versions are kept in a family of products, ensuring maintenance.

A SPL is composed by three essential activities: Core Asset Development, Product Development and Organizational Management.

Core asset development main objective is to establish a production capability for products [26, 27]. It consists of three main outputs [26]: the product line scope, with the definition of the products that constitute the product line; the core assets, that are the basis for production of products in the product line; and the production plan, that is the description of how the products are produced from the core assets.

The product development essential activity depends on the outputs of the core assets development activity and on the specific requirements for each product. Those inputs to this activity if carefully followed will allow the creation of a properly developed product. They must take into consideration the variation points being selected for a given product. This activity may lead to changes in the product line scope definition, generating new core assets and then causing the production plan to be updated.

The Organizational Management essential activity encompasses the need for a strong management commitment that allows the product line to succeed. The role of the organizational management is to determine the funding that will ensure the development of the core assets, to organize and coordinate the activities of core assets development and product development and to contribute with artifacts needed to developing products, like schedules or budgets.

III. RESEARCH OBJECTIVES AND APPROACH

As a main research objective it has been defined the formalization of an approach that combines multi-stage (time-variant stages), with ontological support and multi-level primitives (abstraction levels) for the insurance domain software process development. Multi-stage allows the mapping of multi-level primitives into the stages. The insurance ontological reference is able to unify that mapping. Stages are also useful to delimiter borders and make artifacts as deliverables between stages [28]. Each stage can be viewed as a separate company, delivering products to the next stage and receiving what they need from the previous ones. Each one has its one reality, characteristics and support, all disconnected from the others. This way we can only pass artifacts and documentation between stages, allowing, for instance, the software house to actively communicate with the insurance company and vice-versa.

Taking into account that this work will adopt the insurance domain as the target for experimenting the up to be proposed approach, it will be attempted to formalize a draft of an insurance ontology. This draft will be used along the entire thesis as a semantic reference to support the validation of the multi-stage framework (Figure 1). The intrinsic

complexity of the adopted domain (insurance) will not allow achieving a full definition, in the current work, since the number of concepts and hierarchies involved and the discussion that arises from the need to remove ambiguity would void our attempt.

The establishment of a relation between ontologies and the software development process will be one of the consequences of formalizing a multi-stage framework with ontology as semantic reference. In this context, mapping levels of abstraction to achieve traceability of artifacts and concepts in the business application domain will be a core concern. Some plug-ins for the software development tools will be prototyped to assess the feasibility of adopting the same ontology to support the representation of business knowledge and the execution of validation of final artifacts.

Ontologies will act as a guideline containing the core business and development concepts required by the mode-driven tools to generate specialized and business validated software artifacts. Those artifacts will be traceable along the development lifecycle, from the core assets thru the final product installation and configuration in the customer. Taking into account that the software development process relies on a product line, assets reusability and categorization would be guaranteed by the links established between the artifacts and the ontology.

It will be proposed a process that aims mapping the inherent temporal dimension in product lines with the artifacts' abstraction levels. That mapping can rely on insurance ontologies to guarantee its coherence and act as a guideline.

To accomplish the chosen research objective, it was chosen to use Action Research [4] in conjunction with Experimental Software Engineering [5]. This method involves both research and intervention [29], being the researcher also a practitioner. This kind of research method can address complex real-life problems and the immediate concerns of practitioners [4], being appropriate for investigating the introduction of technologies into any organization [29, 30]. This can be done in a cyclic way, starting with a problem definition, research a possible solution and then see how successful that solution is, and, if not satisfied with it, repeat the steps [29-31]. The problem is systematically studied and based on theoretical considerations [31], scientific documents [30] and models [32]. In Action Research, participants are also researchers, being enrolled in the process and connected to the initiating researcher. According to Villiers [29], action research encompasses action outcomes and research outcomes. As it starts with the problem identification, it becomes an agent of change when taking action upon that problem. When dealing with information systems, action research proves itself useful [33]. The researcher in actively involved and the direct benefits are for both researcher and organization. Also, the knowledge obtained can be immediately applied by the active and dependent observer, the researcher [30], who seeks to study the process and to promote change in the organization. Theory and practice are together, in a cyclic way. To do so, the researcher is actively involved in a software house that works exclusively several leading insurance companies in its country. The software house will act as a laboratory and the deployment sites (in the software house and eventually in some clients) will be used to test the research achievements.

IV. Current Work and Preliminary Results

In the on-going work, domain of insurance is being investigated. This domain is used as an application baseline, which will first help to extract specific needs coming from the real world, and finally, it will help to evaluate the achieved results. The insurance domain deals with business and technological variability in various aspects. Its business models depend upon product line diversity, commercialization channels, or even governmental or institutional laws. There are many different product lines in this specific domain, namely Life, Health, Group, Intermediary Property and Casualty products. Also, it must be accounted that many back office management activities are executed by stable and large legacy systems. Customers interact with those systems through internet and employees through the intranet of the company [1]. As the information demand increases, it also increases time that takes to provide solutions [34]. This process can often trigger intensive and time consuming background. There is no commercial software system or solution that can deliver such content or reasoning in an automatic way. The weight of legacy applications in today's insurance platforms, the need to supply non-standardized information to customers [35], the insurance market dynamics and the continuous need to adapt systems to the market reality are some of the problems focused on this work. This leads to the need for systematic and rigorous modeling of the insurance domain [36].

The insurance industry activity and especially the software house work that as to be done to support that activity relies on information. If by one side – the insurance company – data is needed, by the other – the software house – data has to be provided. There are many legal requirements that need to be fulfilled, like Portuguese Insurance Institute's (ISP) information requirements [37], or internal management information required to run the insurance company or even auditing or statistic maps. To provide such information, there are many applications that support queries that deliver such data. From there emerges an initial difficulty, that is how to know, from the information extraction responsible point of view, where to extract required information due to the variety of sources.

Our current approach aims at mapping ontologies to the software product line development process in a model-driven approach. Regarding Figure 1, the multi-stage ontological support (horizontally defined) allows the mapping of muli-level primitives (vertically placed) into the stages. The ontological referential is able to unify that mapping. Stages are also useful to delimiter borders and make artifacts as deliverables between stages [28]. Each stage can be viewed as a separate company, delivering products to the next stage and receiving what they need from the previous ones. Each one has its one reality, characteristics and support, all disconnected from the others. This way we can only pass artifacts and documentation between stages.

A stage is directly related to time, being an object that has its own lifecycle. Stages may begin with the initial phase, where there is no core assets defined in the domain level. If there are core assets defined, stages automatically begin in a posterior stage. A level refers to abstraction, being the first level the most concrete, where all artifacts are implemented and executed. The last level is the most abstract, usually the building plan that guides the core assets. Some of the needed transformations will be later formalized using QVT [14].

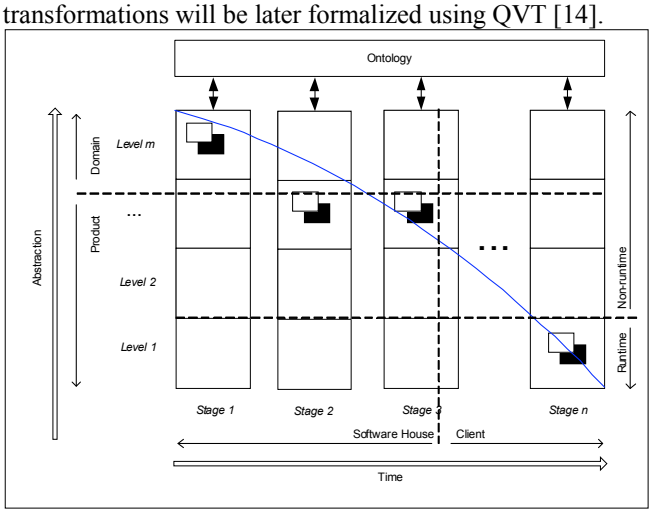

Figure 1: Multi-stage ontologically-sustained approach to SPL multi-level architectures

We have represented the "domain" section, which refers to the domain engineering level or core assets development [26], where the basis for the SPL are developed, in an organized fashion. Also represented is the "product" section, referring the application engineering level, or product development [26] , where the assets previously defined are put together to work.

Two other dimensions are the "client" and the "software house". Regarding the software house, it represents the place where the artifacts are being developed and by client we refer to the customer where the product will be running.

Also related to the previous dimensions we have run-time and non run-time. By non-runtime we mean the levels where the artifact isn't executable. By run-time, it is meant the levels where the artifact is executing.

The chosen approach implies that we need to formalize the multi-stage process approach regarding SPL. This will allow building up the initial representation of the product, regarding an ontological view associated with a temporal dimension of the phases that a system passes from analysis thru run-time. After that, it is necessary to associate a multi-level approach with the multi-stage approach: When dealing with the temporal line defined by the multi-stage, it is possible to adopt several levels of abstraction associated with each phase, thus enabling a better comprehension of the problem we are dealing with. The analysis of the product line viability in technological terms is also a subject requiring attention. We need to match the multi-level and multi-stage product line with technologically-related constraints that may or not exist in a software house. This kind of technological constraints are related with existing frameworks, information repositories, mandatory components, legacy systems, and domain languages, all of them with the ability to frame the levels and/or the stages and also to affect the architecture itself. Finally, the validation of the achieved results with a real case: iteratively test our methodological proposal with a real case, using the software house systems' and product lines' to provide the necessary validation.

V. Work Plan and Implications

To accomplish the previously detailed approach it as been defined a set of activities that will lead to the fulfillment of the research objectives. First and foremost, there is the need to read related approaches and the state of the art in the area. With this literature review it is intended to acquire knowledge about the efforts done in similar problems and approaches. The main areas of study are:

- Role, creation, usage, benefits, critics, advantages of ontologies on information systems;
- Software product lines systems, usage, and adoption, as well as new trends that may appear in that area;
- Model driven engineering/architectures, specially their relation with product-lines and ontologies.

Next, the approach will represent the relation between ontologies and the software development process, always regarding software product lines. A SPL approach to multi-stage software process development will grant us an insight of the full system and help building the ontological system that will grant coherence to all parts. Also there is a temporal dimension inherent to the software process development that may lead us to other findings and research opportunities. We plan to generalize the chosen approach to multi stages in a product line approach. Future constraints to the process, like the time-dimension in modeling [38], or the software product line process formalization [39] will most likely require changes to the model.

Other major step regards the abstraction analysis on the SPL regarding a multi-level view of the line. This approach will also make a valuable contribution for the ontological system. In parallel to this the ontological support will start to appear, shaping itself and guiding features development.

The next phase is the viability analysis of the achieved solution. This phase is mainly a technologically constraint view about the previous research objectives and will impose a strong involvement with a software house development process and methods. It is needed to formalize the multi-stage and the multi-level approaches regarding SPL to the technological reality of the software house. Legacy systems are expected to play a major role in the architecture definition. Other aspects, like official mandatory regulations may imply changes and adaptations to our model.

The last phase encompasses the validation of the achieved results in the previous phases. This will be done by using a real software house's applications as example and test cases. Having enough iterations using action research we expect to prove the viability of our findings in a real situation. The software house's proprietary ERP software will allow us to gather information regarding our model and

the degree of fulfillment of our research objectives. We plan to collect information about our experiments and to validate the previous objective with a real situation.

VI. CONCLUSION

In this paper we related the fundamentals of ontologies, model-driven software development in a software product line process approach. We made an initial proposal of a work plan to tackle an actual and real problem in the software development for highly variable domains (Insurance).

Our proposal regards adding two main dimensions to the product line approach, time and abstraction. If so, the degree of conceptualization rises. A multi-stage process approach is required to deal with the time elapsed in the different product development phase, from core asset thru final product, executed in the customer's environment. The other dimension, the degree of abstraction referred as multi-stage, implies different conceptualizations of the system.

An ontological approach is necessary to unify and guide the process which coordinates all the abstraction and time of the development lifecycle in a product line. Ontological characteristics like the sharing of the same vocabulary [3] makes it suitable for enabling coherence between the phases and the levels. We are trying to create a process that manages the evolution of software product line artifacts over time and ensures the consistent integration of the changes in all affected product line applications. By doing this we expect to manage the product line artifacts in their life spam and thus achieving a possible solution for the highly dynamic insurance domain software development maintainability, conceptualization and configuration problems.

VII. REFERENCES

[1] M. Simonov, et al., "Ontology-driven Natural Language access to Legacy and Web services," *Proc. BIS2004*, 2004.

[2] J. Rilling, et al., "Semantic Technologies in System Maintenance (STSM 2008)," *Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension - Volume 00*, IEEE Computer Society, 2008 of Conference, pp.

[3] D. Gasevic, et al., *Model Driven Architecture and Ontology Development*, Springer, 2006.

[4] D.E. Avison, et al., "Action research," *Commun. ACM*, vol. 42, no. 1, 1999, pp. 94-97.

[5] M.V. Zelkowitz and D.R. Wallace, "Experimental models for validating technology," *Computer*, vol. 31, no. 5, 1998, pp. 23-31.

[6] T. Stahl and M. Völter, *Model-Driven Software Development : Technology, Engineering, Management*, John Wiley & Sons, 2006.

[7] S. Beydeda, et al., eds., Model-Driven Software Development, Springer, 2005.

[8] J. Bézivin, "On the Unification Power of Models," *Proc. Software and System Modeling*.

[9] J.-M. Favre, "Towards a Basic Theory to Model Model Driven Engineering," *Workshop on Software Model Engineering*, 2004.

[10] "OMG - Object Managment Group," 2009; http://www.omg.org.

[11] J. Mukerji and J. Miller, "MDA Guide v1.0.1," 2003; http://www.omg.org.

[12] E. Seidewitz, "What models mean," *Software, IEEE*, vol. 20, no. 5, 2003, pp. 26-32.

[13] K. Czarnecki and S. Helsen, "Classification of Model Transformation Approaches," *Proc. 2nd Workshop on Generative Techniques in the Context of MDA (2003)*.

[14] OMG, "QVT 1.0," 2009; http://www.omg.org/spec/QVT/1.0/.

[15] B. Selic, "The pragmatics of model-driven development," *Software, IEEE*, vol. 20, no. 5, 2003, pp. 19-25.

[16] C. Atkinson and T. Kuhne, "Model-Driven Development: A Metamodeling Foundation," *IEEE Softw.*, vol. 20, no. 5, 2003, pp. 36-41.

[17] T.R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, 1993, pp. 199-220.

[18] O. Corcho, et al., "Ontological Engineering: Principles, Methods, Tools and Languages," *Ontologies for Software Engineering and Software Technology*, 2006, pp. 1-48.

[19] M. Uschold, "Knowledge level modelling: concepts and terminology," *Knowl. Eng. Rev.*, vol. 13, no. 1, 1998, pp. 5-29; DOI http://dx.doi.org/10.1017/S0269888998001040.

[20] P. Wongthongtham, et al., "Software Engineering Ontology for Software Engineering Knowledge Management in Multi-site Software Development Environment," *Proc. 10th International Protégé Conference*, 2007.

[21] Y. Sure, et al., "Why Evaluate Ontology Technologies? Because It Works!," *IEEE Intelligent Systems*, vol. 19, no. 4, 2004, pp. 74-81; DOI http://dx.doi.org/10.1109/MIS.2004.37.

[22] M. Staples and D. Hill, "Experiences adopting software product line development without a product line architecture," *Proc. Software Engineering Conference, 2004. 11th Asia-Pacific*, 2004, pp. 176-183.

[23] F.J.v.d. Linden, et al., *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*, Springer-Verlag New York, Inc., 2007.

[24] K. Pohl, et al., *Software Product-line Engineering – Foundations, Principles, and Techniques*, Springer, 2005.

[25] J. Bosch, *Design and use of software architectures: adopting and evolving a product-line approach*, ACM Press/Addison-Wesley Publishing Co., 2000, p. 354.

[26] P. Clements and L. Northrop, *Software product lines: practices and patterns*, Addison-Wesley, 2002, p. 608.

[27] L.M. Northrop, "SEI's Software Product Line Tenets," *IEEE Softw.*, vol. 19, no. 4, 2002, pp. 32-40.

[28] A. Bragança and R.J. Machado, "Transformation Patterns for Multi-staged Model Driven Software Development," *12th International Software Product Line Conference - SPLC 2008*, IEEE Computer Society Press, Los Alamitos, California, U.S.A., 2008 of Conference, pp. 329-338.

[29] M.R.d. Villiers, "Three approaches as pillars for interpretive information systems research: development research, action research and grounded theory," *Proceedings of the 2005 annual research conference of the SAICSIT on IT research in developing countries*, SAICSIT, 2005 of Conference, pp. 142-151.

[30] R.L. Baskerville, "Investigating information systems with action research," *Commun. AIS*, vol. 2, no. 3es, 1999, pp. 4.

[31] R. O'Brien, "An Overview of the Methodological Approach of Action Research," 1998; http://www.web.net/~robrien/papers/arfinal.html.

[32] V.R. Basili, et al., "Experimentation in Software Engineering," *IEEE Trans. Software Eng.*, vol. 12, no. 7, 1986, pp. 733-743.

[33] M.D. Myers, "Qualitative Research in Information Systems," *MIS Quarterly*, vol. 21, no. 2, 1997, pp. 241-242.

[34] A. Chris van, *Organizational Principles for Multi-Agent Architectures (Whitestein Series in Software Agent Technologies)*, Birkhauser, 2005.

[35] B. Yildiz and S. Miksch, "ontoX - A Method for Ontology-Driven Information Extraction," *Computational Science and Its Applications – ICCSA 2007*, 2007, pp. 660-673.

[36] D. Oberle, et al., "Towards ontologies for formalizing modularization and communication in large software systems," *Applied Ontology*, vol. 1, no. 2, 2006, pp. 163-202.

[37] *Report de Informação para Efeito de Supervisão - Empresas de Seguros*, Instituto de Seguros de Portugal - Norma Regulamentar N.º 21/2003-R, 2003.

[38] C.I. Theodoulidis and P. Loucopoulos, "The time dimension in conceptual modelling," *Inf. Syst.*, vol. 16, no. 3, 1991, pp. 273-300.

[39] J. Bosch, "Adopting Software Product Lines: Approaches, Artefacts and Organization," *Proc. Proceedings of the International Workshop on Product Line Engineering - The Early Steps: Planning, Modeling, and Managing (PLEES'01)*, IESE-Report No. 050.01/E, 2001.