Modelling Variability, Evolvability, and Adaptability in Service Computing (a vision for future research)

Modelling Variability, Evolvability, and Adaptability in Service Computing

(a vision for future research)

M.H. ter Beek, S. Gnesi
ISTI–CNR, Pisa, Italy
{terbeek,gnesi}@isti.cnr.it

A. Fantechi
Università di Firenze, Italy
fantechi@dsi.unifi.it

G. Zavattaro
Università di Bologna, Italy
zavattar@cs.unibo.it

***Abstract*—We present a vision for future research on an emerging topic in software engineering, namely the synergy between Software Product Line Engineering (SPLE) and Service-Oriented Computing (SOC). Our aim is to develop rigorous modelling techniques and analysis and verification tools that can be used for the systematic, large-scale provision and market segmentation of *software services*. We foresee flexible design techniques with which *software service line organizations* can develop novel classes of service-oriented applications that can easily be adapted to customer requirements as well as to changes in the context in which, and while, they execute. By superposing variability mechanisms on current languages for service design, based on policies and strategies defined by service providers, we envision the possibility to identify variability points that can be triggered at run time to increase adaptability and optimize the (re)use of resources.**

I. Introduction

Product Line Engineering (PLE) is a paradigm to develop a family of products using a common platform and mass customization [34]. This engineering approach aims to lower production costs of the individual products by letting them share an overall reference model of the product family, while at the same time allowing them to differ with respect to particular characteristics in order to serve, e.g., different markets. As a result, the production process in PLE is organized so as to maximize commonalities of the products and at the same time minimize the cost of variations. The product variants can be derived from the product family, which allows reuse and differentiation of products of the family. Software Product Line Engineering (SPLE) is a paradigm for developing a diversity of software products and software-intensive systems based on the underlying architecture of an organization's product platform [11], [38].

Service-Oriented Computing (SOC) has emerged as an evolutionary new paradigm for distributed and object-oriented computing [37], [39]. Services are autonomous, distributed, and platform-independent computational elements capable of solving specific tasks (ranging from answers to simple requests to complex business processes) which all need to be described, published, categorized, discovered, and then dynamically and loosely coupled in novel ways (composed, orchestrated) so as to create largely distributed, interoperable, and dynamic applications and business processes which span organizational boundaries as well as computing platforms. In the end, service-oriented systems deliver application functionalities as services to either end-user applications or other services. Their underlying infrastructures are called Service-Oriented Architectures (SOAs). Unlike any earlier computing paradigm, SOC is destined to exert a continuous influence on modern day domains like e-Commerce, e-Government, e-Health, and e-Learning.

In this paper, we present a vision for future research on an emerging topic in software engineering, namely the synergy between SPLE and SOC [6], [12], [20], [22]–[27], [29], [40]. Our aim is to develop rigorous modelling techniques as well as analysis and verification tools that can be used for the systematic, large-scale provision and market segmentation of *software services*. We foresee flexible design techniques with which *software service line organizations* can develop novel classes of service-oriented applications that can easily be adapted to customer requirements as well as to changes in the context in which, and while, they execute. By superposing variability mechanisms on current languages for service design, based on policies and strategies defined by service providers, we envision the possibility to identify variability points that can be triggered at run-time to increase adaptability and optimize the (re)use of resources. The resulting design techniques and support tools will be able to assist organizations to plan, optimize, and control the quality of software service provision, both at design- and at run-time. We currently do not aim to assist also the early requirements engineering phase of system modelling; instead, the reader is referred to [20] and its references for work on enhancing the i^* framework with variability modelling capabilities.

Our concrete proposal is to first focus on the definition of the formal modelling framework, with a threefold objective:

1) the extension of (semi)formal existing notations and languages for SOC with notions of variability through which increased levels of flexibility and adaptability can be achieved in software service provision;
2) the definition of a rigorous semantics of variability over behavioural models of services that can support a number of design- and run-time analysis techniques;
3) the development of analysis and verification techniques that remain effective over specifications with variability points, including situations in which the variability is triggered at run-time.

As a motivating example, consider a case study drawn from daily academic life, namely a software system that supports the teaching activities of a university. This system is constituted by a set of services offered to the teachers, to the students and to the university staff, following a SOC paradigm. Migration of such a system to a different university will introduce some changes due to the different governance structure, different size, and/or different teaching organization.

A product family, or product line, can therefore be envisaged to encompass all possible variants. Hence, a definition of a service-oriented product line is needed. This product line definition should be able to cope with static variability, which is confined to deal with the difference between universities. However, the evolution of laws introduce another form of variability, namely dynamic variability, which needs to be addressed in order to make the system adaptable to the evolution of its environment.

II. Beyond the State of the Art

The SPLE and SOC approaches to software development share a common goal: Both encourage an organization to reuse existing assets and capabilities rather than repeatedly redevelop them for new software systems. These approaches enable organizations to capitalize on reuse to achieve desired benefits such as productivity gains, decreased development costs, improved time-to-market, higher reliability, and competitive advantage. Their distinct goals may be stated as follows [40].

- SPLE Systematically capture and exploit commonalities among a set of related systems, while managing variations for specific customers or market segments;
- SOC Enable the assembly, orchestration, and maintenance of enterprise solutions to quickly react to changing business requirements.

Contributions concerning the connection between SPLE and SOC are starting to emerge in the software engineering community [6], [20], [22]–[25], [29] and a recent workshop series [12], [26], [27] examines the connection between SOA and SPL approaches with the purpose of answering how the two techniques can benefit from each other. Indeed service-oriented systems can benefit from SPL's variation management approaches to identify and design services targeted to multiple service-oriented systems.

In [6] an approach is proposed to transfer the main peculiarities of a SPL (i.e., asset reuse and variation mechanisms) to service-oriented systems development, in order to realize a service-oriented systems line. In this way, a method is provided to easily adapt a service-oriented application to different customer needs in changeable environments. The service-oriented systems line consists of two main phases using respectively the Business Process Lines concept, that allows realizing a process variant specifically for the given requirements and the Process-Oriented Development paradigm that allows automating this model and transforming it into a service-oriented system.

In [24] the application of dynamic product line practices is proposed to facilitate the design of service-based systems. Atomic services are used to represent basic system features. A composition of such services creates a configuration, which is a product of the product line. The requirements of an application are modelled in terms of a feature diagram during a domain analysis phase, while distinguishing between atomic and composite features. The entire system is built from atomic features, mapped directly onto a set of existing services.

In [29] feature analysis and service analysis are combined into a method to guide developers to developed service-oriented product lines. While the methods inherits the flexibility from service orientation, it still allows to manage variability through SPLE techniques. It comes with an architectural model for the systematic development of service-oriented product lines, and with support for dynamic reconfiguration.

We envision to go beyond the state-of-the-art of the connection between SPLE and SOC, with a twofold objective:

1) to provide a full formal model of such connections, with verification techniques based upon them;
2) to address run-time adaptability by extending the scope of the flexibility that can be achieved by introducing variability in service definitions.

III. Overall Strategy

The primary objective of the research we envision is to add variability and adaptability to the principles of SOC. The strategy is, in the first instance, to inherit from SPLE mechanisms to include variability notions in a software artifact. A crosscutting concern is to guarantee basic correctness assumptions of the provided services, in terms of certain desired qualitative and quantitative properties, by means of formal modelling of variability and adaptability. The rationale is that families of services should be formally defined with a service-oriented description language such as, e.g, UML4SOA (www.uml4soa.eu), BPEL 2.0 (www.oasis-open.org/committees/wsbpel) and/or JOLIE (www.jolie-lang.org), suitably extended to deal with variability and adaptability. Current service-oriented languages do not support (or support in a very limited form) the possibility to *configure*, *adapt*, and *reconfigure* the specified system. Subsequently, the extension along the same lines of the formal verification techniques already available for such languages, will provide the possibility to conduct formal property verification on service-oriented specifications that include variability.

In our vision, variability needs to be investigated from several points of view:

1) The formal modelling of variability and evolvability;
2) The linguistic mechanisms to express variability in service-oriented descriptions;
3) The development of formal verification techniques and tools for service-oriented systems utilizing variability.

We now provide details on how to pursue these three issues.

A. Modelling Variability and Evolvability

To address the aforementioned crosscutting concern on correctness, we need to provide a formal model of service lines that is able to capture the notion of variability, i.e., it is able to express the variations that certain characteristics of services can be subject to. To this aim, the basics of variability modelling need to be investigated in connection with the formal modelling of SOAs.

A common reference model for the behaviour of a service in SOC computing is based on Labelled Transition Systems (LTSs): A service is defined as a state machine that interacts with its clients and with other services, and such interactions may trigger a transition to a different state.

An extension of LTSs, called Modal Transition Systems (MTSs), has been proposed as a formal model for product families [19], [28], allowing one to embed in a single model the behaviour of a family of products that share the basic structure of states and transitions, transitions which can moreover be seen as mandatory or possible for the products of the family. In [17], the MTS concept has been pushed to a more general form, allowing more precise modelling of the different kinds of variability that can typically be found in the definition of a product family.

On the other hand, deontic logics [1], [30], [32], [33], [41] have recently become popular in computer science for formalizing descriptional and behavioural aspects of systems. This is mainly because they provide a natural way to formalize concepts like violation, obligation, permission, and prohibition. Intuitively, they permit one to distinguish between correct (normative) states and actions on the one hand and non-compliant states and actions on the other hand. This makes deontic logic a natural candidate for expressing the conformance of members of a family of products with respect to variability rules.

A first goal is therefore to establish a common reference model that develops these concepts to fully take into account the peculiarities of SOC and the characteristics of SPLE. When the modelling of variability in SOC is consolidated, the reference model will be extended to address run-time adaptability. However, this is a challenging task that can only be attacked when the modelling of variability in SOC is fully understood (cf. Sect. III-C).

A second, future goal is to model not only functional variability but also quality attribute variability in SPLE [16]. In particular, it would be interesting to study how to model and maintain certain Quality of Service (QoS) levels and QoS-aware service composition in the presence of variability, evolvability, and adaptability [31], [36].

B. Extending Service-Oriented Languages

The idea is to investigate the extensions/modifications that need to be applied to current service-oriented languages in order to support both static and dynamic variability. It is necessary to revisit choreography, orchestration, and behavioural contract languages.

Concerning static variability, we need to investigate the most effective ways to include constructs in the languages to express so-called *variation points*, i.e., the points where a single service, or a choreography, or an orchestration, admit different variants. The choice of variants is made at configuration time (i.e., product derivation time), hence we expect that a single product respects the syntax of the considered language.

A particular role can be played in this regard by behavioural contracts. Behavioural contracts have been independently introduced in SOC by various authors (see, e.g., [7]–[9] and the references therein), and represent an abstraction of the expected behaviour of a service according to its main observable features: a notion of compliance is also defined, to verify that a service actually presents the abstraction required by the contract. Indeed, the very same definition of contract entails a definition of a family of services: the family of all services that are compliant to the contract. Hence, one of the possible ways to express variability is just an invocation to a service compliant to a contract, leaving free the binding to an actual service at configuration time.

Regarding dynamic variability, several directions of investigation can be pursued. For instance, we plan to study two different scenarios: *predictable* and *unpredictable* updates. In general, a dynamic update is triggered by some specific events indicating the necessity to modify the system. If the modifications to be applied are known at design-time (i.e., they are predictable), then it is possible to program such modifications using mechanisms similar to fault-handling in standard languages. On the other hand, if the modifications to be applied to the system are not known at design-time (i.e., they are unpredictable), then it is not only necessary to extend the language, but the system architecture needs to be changed by adding specific components that we call *reconfiguration managers*.

First of all, we can address it by including in the language *evolution hooks*, i.e., information on part of the system structure or behaviour on which modifications could be applied, allowing the programmer specify points that could be affected by future system reconfigurations. A reconfiguration manager is responsible for catching the events indicating the necessity to reconfigure the system, and then it reacts by applying the required modifications to the evolution hooks. In our vision, a reconfiguration manager should follow some update rules that can be dynamically modified, thus giving the possibility to inject in the system new adaptation policies that were unknown at design-time.

The dynamic variability logic should be developed separately, e.g., as a set of evolution rules. Such rules could be created/changed after the application has been deployed without affecting the running application. Evolution should be enacted by an evolution manager, possibly composed by different evolution servers. At run-time, such servers should check the environment conditions and the user needs, control whether some modification has to be applied to the application, and exploit the evolution hooks provided by the application to reconfigure it.

Based on the extended languages, also the theories already developed for choreographies, orchestrators, and behavioural contracts need to be revisited. These three distinct aspects of SOC are strictly related. For instance, one could extract from choreographies the behavioural contracts of the involved services, or verify whether or not an orchestrator respects a behavioural contract. The theories relating these aspects need to be revisited in order to cope with variability.

In particular, variability of the system architecture can be more easily expressed with choreography languages. In the case of dynamic variability, in order for the reconfiguration to take place, the modifications must be applied to the running services. Therefore, appropriate projection functions must be defined which can automatically obtain, from the variability expressed with choreography languages, the modifications to be applied to the evolution hooks of the relevant services. Moreover, current behavioural contract theories have to be enhanced in order to include also dynamic checks that are able to verify, at reconfiguration time, whether or not the modified system still preserves some expected properties such as service compliance.

These proposed enhancements to SOC languages have to be validated by experimenting them on an existing SOC framework, like JOLIE (www.jolie-lang.org). JOLIE is a fully fledged orchestration language that already includes some form of dynamic variability such as the possibility to rebind service ports or replace internal services. The goal in this case is to extend JOLIE with evolution hooks, to define a choreography language for JOLIE applications, and to assess the developed behavioural contract theories by applying them to this specific JOLIE-based framework.

C. Develop Verification Techniques/Tools

The combination and extension of the ideas underlying the modelling and verification techniques and tools that have been developed in the SOC domain with those from the SPLE domain, will have the aim of developing analysis and verification techniques that support design-time verification and validation, run-time monitoring, and verification of flexible and adaptable services. The fact that the resulting analysis and verification techniques should still be effective over specifications with variability points, including situations of variability triggered at run-time, requires particular care.

A first concern is the analysis of abstract properties (qualitative and possibly quantitative) both at the level of the family specification and at that of their derived products. Exemplary qualitative properties of services are [18]:

Availability: a service is always capable of accepting a request;

Responsiveness: a service guarantees a response to each received request;

Reliability: a service guarantees a successful response to each received request.

Quantitative properties instead include QoS properties based on a notion such as cost, as well as classical quantitative properties, stating that certain properties hold within a given probability bound. Recent results on the verification of such quantitative and qualitative properties on service descriptions [4], [5], [13]–[15], [18] can be adapted in order to be able to deal not only with single services, but directly with service family specifications. In this way we can factorize both the time and the cost that is needed to verify products that have been correctly derived from a family definition.

A second, related, concern will be the adoption of specific analysis and verification techniques aimed at proving correct derivations of products from a service family definition [2]. We have already mentioned (in Sect. 3.1) the recent interest raised by deontic logics for the modelling of variability. Indeed, some work [2], [3] has been done on the use of deontic-style logics for modelling notions of variability in product family descriptions, in two different directions: characterizing feature models by direct modelling of constraints over the products of a family, and proposing behavioural extensions of deontic-style logics. A behavioural extension of a deontic-style logic permits one to express in a unique framework both behavioural aspects, using standard branching-time logic operators, and static constraints over the products of a family (usually a separate expression in a first-order logic is required). Proper variants of established model checkers, like CMC (fmt.isti.cnr.it/cmc), must be defined as automatic verification tools for checking such properties.

A third concern is the introduction of run-time adaptability, which presents a big challenge for the off-line verification by model checking [10]. This question requires innovative verification techniques and a deep understanding of the relation between sought service properties and variability in the definition of services. For instance, a typical question can be whether a SOA satisfies a given property, irrespective of which variant has actually been chosen, or of which evolution is occurring inside the architecture. Finally, variability is used differently in adaptive system modelling. For product lines, a particular set of variants is chosen at compile time (or at deployment time) while for an adaptive system variability has to be available and managed also at run-time [35]. A possible way out is thus to represent the product family architecture at run-time [21].

IV. Application Domains

The research activity envisioned in this paper will produce innovative elements to be used in the definition of a design and development methodology for the systematic large-scale production of software systems and their market segmentation. These innovative elements will consist of the definition of techniques for the flexible modelling and design of software adaptability, by means of which it will be possible to develop *services families*. These are new classes of service-based applications, easily derivable from the definition of the *family* through adaptation to client needs or to modification of the specific context for which the application has been deployed.

We can refer to the motivating example already introduced in Sect. 1, i.e. a software system that supports the teaching activities of a university. This system is constituted by a set of services offered to the teachers, to the students and to the university staff, following a SOC paradigm. Selling such a system to different universities requires to manage variability due to the different governance structure, different size, and/or different teaching organization of the different universities. A product family, or product line, can therefore be envisaged to encompass all possible variants. Hence, a definition of a service-oriented product line is needed. This product line definition is however confined to cope with static variability, i.e., differences between universities. However, the rapid evolution of laws (at least in Italy...) and their following reception from the university offices can be dealt efficiently only introducing dynamic adaptability and evolvability.

Another application domain where we can expect that the proposed advancements can have a favourable impact is that of the so-called *package-based software distributions*, an approach adopted, typically, by FOSS (Free and Open-Source Software) to flexibly manage the different variants and versions of a software system, as exemplified by the distribution of open-source operating systems like LINUX.

One of the main challenges for such distributions is the possibility to scale the system to distributed computing platforms, beyond the bound of executing on a single computing element. In the framework of recent virtualization technologies known as Cloud Computing, e.g., the management of virtual machines that support flexible and on-demand computing resource offering, such as "Infrastructures as a Service" (IaaS) and "Platform as a Service" (PaaS) turns to be extremely complex.

Traditionally, the package-based software distributions provide flexible tools for personalizing and updating the system. Moreover, they provide tools that support the system administrator to design and manage the system. Such tools are based on a complex interdependency network between packages that allow the automatic installation of updates and of new system components. Normally, as the system executes on a single computing element, such elements should be switched off first to make new functions available (cold update). This approach turns out to be impossible in a distributed system. A new approach (hot update), which does not interrupt the system functioning, is needed.

The definition of mechanisms supporting the update of only the relevant components is fundamental for the widespread utilization of emerging technologies such as, e.g., Cloud Computing. This technology, based on virtualization, is at the basis of the so-called *disappearing computer* scenario, according to which computing and storage devices are physically moved far from the end user, and are offered by specific vendors able to deliver computing and storage power as a remote service (see, e.g., Google AppEngine and Amazon EC2).

Concerning remote file storage, some services already reached large diffusion, see, e.g., Dropbox (that provides storage, sharing and synchronization of files) that in February 2010 already counted more than 4 million users. One of the aspects that contributed most to the success of Dropbox is the possibility to provide the user with a virtual *hard disk* shared by different devices. In this way, this service realized the *Forever Yours* model — one of the objectives of the European Commission FP7 in ICT — at least as far as data and file storage is concerned. Cloud Computing aims at the realization of this model also as far as programs and computing power is concerned. Nevertheless, such systems are subject to an evolution which is far more complex than that in file systems (in which only creation, change, and delete operations can be performed) as strong dependencies exist among the involved software components.

The techniques developed according to the research vision outlined in this paper will be applicable in this specific field, with the aim of facilitating the management of this kind of services, making the user configuration completely automatic, and providing the user with tools supporting the update and evolution of the system.

One can think also of many other fields of applications, in which a high degree of configurability and an easy and fast adaptation are extremely critical. One could consider, e.g., e-Health where hospital services must be highly flexible in order to quickly adapt to specific medical needs, or the emergency management field where the support tools must be quickly exploitable in an always different and unstable context, in which run-time adaptability features are clearly needed.

In all these application fields, the models, techniques and tools developed according to the vision proposed in this paper can achieve significant advantages in terms of development costs, as it will be able to factorize common elements as well as the verification processes.

REFERENCES

[1] L. Åqvist, Deontic Logic. In *Handbook of Philosophical Logic, 2nd Edition* (D. Gabbay and F. Guenthner, Eds.), Volume 8, Kluwer Academic, 2002, 147–264.

[2] P. Asirelli, M.H. ter Beek, A. Fantechi and S. Gnesi, Deontic Logics for Modeling Behavioural Variability. In *Proceedings of the 3rd International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'09)* (D. Benavides, A. Metzger and U. Eisenecker, Eds.), ICB Research Report 29, Universität Duisburg-Essen, 2009, 71–76.

[3] P. Asirelli, M.H. ter Beek, A. Fantechi and S. Gnesi, A deontic logical framework for modelling product families. In *Proceedings of the 4th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'10)* (D. Benavides, D. Batory and P. Grünbacher, Eds.), ICB Research Report 37, Universität Duisburg-Essen, 2010, 37–44.

[4] M.H. ter Beek, A. Fantechi, S. Gnesi and F. Mazzanti, An action/state-based model-checking approach for the analysis of communication protocols for Service-Oriented Applications. In *Formal Methods for Industrial Critical Systems—Revised Selected Papers of the 12th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'07), Berlin, Germany* (S. Leue and P. Merino, Eds.), LNCS 4916, Springer, 2008, 133–148.

[5] M.H. ter Beek, A. Fantechi, S. Gnesi and F. Mazzanti, A state/event-based model-checking approach for the analysis of abstract system properties. To appear in *Science of Computer Programming*, 2010.

[6] N. Boffoli, M. Cimitile, F. Maria Maggi, G. Visaggio, Managing SOA System Variation through Business Process Lines and Process Oriented Development. In [27].

[7] M. Bravetti and G. Zavattaro, A Theory for Strong Service Compliance. In *Proceedings of the 9th International Conference on Coordination Models and Languages (COORDINATION'07), Paphos, Cyprus* (A.L. Murphy and J. Vitek, Eds.), LNCS 4467, Springer, 2007, 96–112.

[8] M. Bravetti and G. Zavattaro, Contract based Multi-party Service Composition. In *Proceedings of the 1st International Symposium on Fundamentals of Software Engineering (FSEN'07), Tehran, Iran* (F. Arbab and M. Sirjani, Eds.), LNCS 4767, Springer, 2007, 207–222.

[9] M. Bravetti and G. Zavattaro, Towards a Unifying Theory for Choreography Conformance and Contract Compliance. In *Proceedings of the 6th International Symposium on Software Composition (SC'07), Braga, Portugal* (M. Lumpe and W. Vanderperren, Eds.), LNCS 4829, Springer, 2007, 34–50.

[10] E.M. Clarke, O. Grumberg and D.A. Peled, *Model Checking*. The MIT Press, 2000.

[11] P.C. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[12] S.G. Cohen and R.W. Krut (Eds.), *Proceedings of the 1st Workshop on Service-Oriented Architectures and Software Product Lines: What is the Connection? (SOAPL'07)*. Technical Report CMU/SEI-2008-SR-006, 2008.

[13] R. De Nicola, D. Latella and M. Massink, Formal modeling and quantitative analysis of KLAIM-based mobile systems. In *Proceedings of the 20th Annual ACM Symposium on Applied Computing (SAC'05)*, ACM, 2005, 428–435.

[14] R. De Nicola, J.-P. Katoen, D. Latella and M. Massink, Towards a logic for performance and mobility. *Electronic Notes in Theoretical Computer Science* 153, 2 (2006), 161–175.

[15] R. De Nicola, J.-P. Katoen, D. Latella, M. Loreti and M. Massink, Model checking mobile stochastic logic. *Theoretical Computer Science* 382, 1 (2007), 42–70.

[16] L. Etxeberria, G. Sagardui and L. Belategi, Modelling Variation in Quality Attributes. In *Proceedings of the 1st International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'07)* (K. Pohl, P. Heymans, K.C. Kang and A. Metzger, Eds.), Lero Technical Report 2007-01, 2007, 51–59.

[17] A. Fantechi and S. Gnesi, Formal modelling for Product Families Engineering. In *Proceedings of the 12th International Software Product Lines Conference (SPLC'08)*, IEEE, 2008, 193–202.

[18] A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese and F. Tiezzi, A Model Checking Approach for Verifying COWS Specifications. In *Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering (FASE'08)* (J.L. Fiadeiro and P. Inverardi, Eds.), LNCS 4961, Springer, 2008, 230–245.

[19] D. Fischbein, S. Uchitel and V.A. Braberman, A Foundation for Behavioural Conformance in Software Product Line Architectures. In *Proceedings of the 2nd Workshop on the Role of Software Architecture for Testing and Analysis (ROSATEA'06)* (R.M. Hierons and H. Muccini, Eds.), ACM, 2006, 39–48.

[20] P. Grünbacher, D. Dhungana, N. Seyff, M. Quintus, R. Clotet, X. Franch, L. López and J. Marco, Goal and Variability Modeling for Service-oriented System: Integrating i^* with Decision Models. In *Proceedings of Software and Services Variability Management Workshop – Concepts Models and Tools* (T. Männistö, E. Niemelä and M. Raatikainen, Eds.), Helsinki University of Technology, Software Business and Engineering Institute Research Reports 3, HUT-SoberIT-A3, 2007, 99–104.

[21] S.O. Hallsteinsen, E. Stav, A. Solberg and J. Floch, Using Product Line Techniques to Build Adaptive Systems. In *Proceedings of the 10th International Software Product Lines Conference (SPLC'06)*, IEEE, 2006, 141–150.

[22] A. Helferich, G. Herzwurm and S. Jesse, Software Product Lines and Service-Oriented Architecture: A Systematic Comparison of Two Concepts. In [12].

[23] A. Helferich, G. Herzwurm, S. Jesse and M. Mikusz, Software Product Lines, Service-Oriented Architecture and Frameworks: Worlds Apart or Ideal Partners? In *Trends in Enterprise Application Architecture*, LNCS 4473, Springer, 2007, 187–201.

[24] P. Istoan, G. Nain, G. Perrouin and J.-M. Jezequel, Dynamic Software Product Lines for Service-Based Systems. In *IEEE 9th International Conference on Computer and Information Technology*, IEEE, 2009.

[25] M. Koning, C.-a. Sun, M. Sinnema and P. Avgeriou, VxBPEL: Supporting variability for Web services in BPEL. *Information and Software Technology* 51 (2009), 258–269.

[26] R.W. Krut and S.G. Cohen (Eds.), *Proceedings of the 2nd Workshop on Service-Oriented Architectures and Software Product Lines: Putting Both Together (SOAPL'08)*. In *Proceedings of the 12th International Software Product Lines Conference (SPLC'08), Second Volume (Workshops)* (S. Thiel and K. Pohl, Eds.), Lero, University of Limerick, Ireland, 2008, 115–147.

[27] R.W. Krut and S.G. Cohen (Eds.), *3rd Workshop on Service-Oriented Architectures and Software Product Lines: Enhancing Variation (SOAPL'09)*. In *Proceedings of the 13th International Software Product Lines Conference (SPLC'09)* (D. Muthig and J.D. McGregor, Eds.), ACM, 2009, 301–302.

[28] K.G. Larsen, U. Nyman and A. Wąsowski, Modal I/O Automata for Interface and Product Line Theories. In *Proceedings of the 16th European Symposium on Programming Languages and Systems (ESOP'07)* (R. De Nicola, Ed.), LNCS 4421, Springer, 2007, 64–79.

[29] J. Lee, D. Muthig and M. Naab, An Approach for Developing Service Oriented Product Lines. In *Proceedings of the 12th International Software Product Lines Conference (SPLC'08)*, IEEE, 2008, 275–284.

[30] A. Lomuscio and M.J. Sergot, Deontic interpreted systems. *Studia Logica* 75, 1 (2003), 63–92.

[31] N.B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas and V. Issarny, QoS-Aware Service Composition in Dynamic Service Oriented Environments. In *Proceedings of the 10th International Middleware Conference (Middleware'09)* (J.M. Bacon and B.F. Cooper, Eds.), LNCS 5896, Springer, 2009, 123–142.

[32] L. McCarty, Permissions and Obligations. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI'83)* (A. Bundy, Ed.), William Kaufmann, 1983, 287–294.

[33] J.-J.Ch. Meyer and R.J. Wieringa (Eds.), *Deontic Logic in Computer Science: Normative System Specification*, Wiley, 1993.

[34] M.H. Meyer and A.P. Lehnerd, *The Power of Product Platforms: Building Value and Cost Leadership*. The Free Press, 1997.

[35] B. Morin, O. Barais, J.M. Jézéquel, F. Fleurey and A. Solberg, Models Run.time to Support Dynamic Adaptation. *IEEE Computer* 42, 10 (2009), 44–51.

[36] A.D. Mosincat, W. Binder and M. Jazayeri, Dynamically Adaptive Systems through Automated Model Evolution Using Service Compositions. In *Proceedings of the 9th International Conference on Software Composition (SC'10)* (B. Baudry and E. Wohlstadter, Eds.), LNCS 6144, Springer, 2010, 82–89.

[37] M.P. Papazoglou, P. Traverso, S. Dustdar and F. Leymann, Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer* 40, 11 (2007), 38–45.

[38] K. Pohl, G. Böckle and F. van der Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[39] M.P. Singh and M.N. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*. Wiley, 2005.

[40] Ch. Wienands, Studying the Common Problems with Service-Oriented Architecture and Software Product Lines. Presented at the *4th Service-Oriented Architecture (SOA) & Web Services Conference*, 2006.

[41] R. Wieringa, J.-J.Ch. Meyer and H. Weigand, Specifying Dynamic and Deontic Integrity Constraints. *Data and Knowledge Engineering* 4 (1989), 157–189.