Variability and Rigour in Service Computing Engineering

Maurice H. ter Beek, Stefania Gnesi
Istituto di Scienza e Tecnologie dell'Informazione
ISTI–CNR, Pisa, Italy
Email: {terbeek,gnesi}@isti.cnr.it

Alessandro Fantechi
DSI, University of Florence, Italy
ISTI–CNR, Pisa, Italy
Email: fantechi@dsi.unifi.it

José L. Fiadeiro
Department of Computer Science
University of Leicester, UK
Email: jose@mcs.le.ac.uk

***Abstract*—We present a research agenda on an emerging topic in software engineering, namely the synergy between Software Product Line Engineering (SPLE) and Service-Oriented Computing (SOC). Our proposal is to develop rigorous modelling techniques as well as analysis and verification support tools for assisting organisations to plan, optimise, and control the quality of *software service* provision, both at design time and at run time. We foresee a flexible engineering methodology according to which *software service line organisations* 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 engineering, based on policies and strategies defined by service providers, we envision the possibility of identifying variability points that can be triggered at run time to increase adaptability and optimise the (re)use of resources.**

I. Introduction

Product Line Engineering (PLE) is an approach to develop a family of products using a common platform and mass customisation [27]. 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 organised so as to maximise commonalities of the products and at the same time minimise 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 organisation's product platform [12], [29]. Variability among products is made explicit by variation points, i.e., places in design artifacts where a specific decision is reduced to several features but the feature to be chosen for a particular product is left open (like optional, mandatory, or alternative features). Variety from a single product platform is achieved by identifying such variability points. Variability management is the key aspect differentiating SPLE from 'conventional' software engineering.

Research funded by the project D-ASAP (MIUR–PRIN 2007) and by the RSTL project XXL of the National Research Council of Italy (CNR).

Service-Oriented Computing (SOC) is a paradigm for distributed computing [28], [30]. Services are autonomous, distributed, and platform-independent computational elements capable of solving specific tasks (ranging from answering simple requests to complex business processes) which all need to be described, published, categorised, discovered, and then dynamically and loosely coupled in novel ways (composed, orchestrated). They can be used to create largely distributed, interoperable, and dynamic applications and business processes which span organisational boundaries as well as computing platforms. In the end, SOC systems deliver application functionalities as services to either end-user applications or other services. Service-Oriented Architectures (SOAs) result from applying service orientation. SOC is now maturing as a research discipline, and is currently supported by two main technologies — Web Services (WSs) and Grid Computing — which exploit the pervasiveness of the Internet. Unlike any earlier computing paradigm, SOC is destined to exert a continuous influence on modern day domains like e-Commerce, e-Government, and e-Health.

Current software engineering methodologies are not yet at a level where the full potential of SOC can be realised. From a software engineering point of view, SOC brings to the front aspects that have been addressed in the context of component-based development, but there are differences that require rethinking the processes through which software services are developed. First of all, the typical 'static' scenario in which components are assembled to build a (more or less complex) system that is delivered to a customer no longer applies. Instead, SOC supports more 'dynamic' development scenarios in which (smaller) applications are developed to run on global computers like the Internet and respond to business needs by interacting with services and resources that are globally available.

Developing services that can be discovered and bound to, at run time, by the middleware (not by skilled engineers at design time) is not the same as developing bespoke software applications or adapting existing software to a given customer's set of requirements. In particular, service providers need forms of assembly that offer new levels of flexibility and adaptability to match the functionalities required by client applications as they execute and negotiate Service Level Agreements (SLAs) that can be monitored and acted upon at run time to ensure Quality of Service (QoS). Likewise, new levels of adaptability are required to ensure resilience w.r.t. faults and changes in the context in which service instances are executing, while they are executing.

The fact that the selection of services is not made from a fixed universe of components but through run-time discovery, gives rise to an open and dynamic 'market' of service provision that operates differently from the traditional one of software components. That is, engineering service provision is quite different from, and more demanding than, that of component-based development. Therefore, SOC requires the organisational context in which services are developed and evolved to be explicitly modelled in terms of constraints and policies that control the usage of resources and the establishment of business partnerships, much in line with what are often called 'virtual organisations', and support to be provided for analysis and monitoring of run-time provision.

Another important factor in service computing engineering is the level of rigour that is required for service description, as matchmaking and binding need to be performed by middleware. Although progress has been made towards formal foundations of SOC, there is still a significant gap between practice (programming) and theory (formal methods and analysis tools). One of the reasons is that the necessity to address the many issues that arise in SOC (such as concurrent activities, asynchronous interactions, workflow coordination, business transactions, resource usage, security) has produced a large number of languages, techniques, and tools. Although consensus has begun to distil around some of these, there is not yet a broadly accepted service computing engineering platform that developers can use. Currently, a producer of service applications must cope with the difficulty of choosing a particular framework, method, or language and, therefore, it is inevitable that some form of rigidity is introduced in the engineering process, which does not allow for prompt response to different market demands.

In this paper, we present a research agenda on the synergy between SPLE and SOC [6], [10], [13], [18], [19], [21]–[25], [31]. Our aim is to develop an engineering methodology for systematic, large-scale provision and market segmentation of *software services*, based on rigorous modelling, analysis, and verification techniques. Initially, we propose to focus on defining the formal framework, with a threefold objective:

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

As a motivating example, consider a software company that developed a package to be sold to those interested in starting a travel agency service (e.g. on the web). This company provides a choice among a family of products with different prices and different features. All products provide the service features *hotel*, *flight*, and *train* reservation: the coordination component uses *predefined* external services (one for each business sector) to retrieve a list of quotes. These products can be enhanced in two ways:

1) By adding as *alternative* feature the possibility to choose, only for flights and hotels, from *multiple* external services in order to retrieve the best quotes through more than one service. This requires that proper coordination addresses more hotel and flight services, and proper data fusion is done with the data received by the contacted external services;
2) By adding as *optional* feature the possibility for a customer to book a *leisure tour* during his/her stay at a hotel. This is achieved through an additional component that interacts with an external leisure tour service. However, as the provided tour packages may include a hotel in a different location for a subset of nights, the reservation of a tour *requires* an interaction with the hotel service to offer a feature that allows to *cancel part* of the room reservations at the main hotel location for such nights. A variant of the coordination model can do so.

When combined, these additional features lead to the eight different products listed in Table I.

II. State of the Art

The SPLE and SOC approaches to software development share a common goal: Encourage an organisation to reuse existing assets and capabilities rather than repeatedly redevelop them for new software systems. Both thus enable organisations to capitalise 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 [31].

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], [10], [18], [19], [21], [22], [25] and a recent workshop series [13], [23], [24] has examined the connection between SOA and SPL approaches by studying how the two techniques can mutually benefit from each other. For instance, SOC systems can benefit from SPL's

Table I
Valid products of the Travel Agency product family

features		variability	products							
			1	2	3	4	5	6	7	8
train reservation	predefined services	mandatory	√	√	√	√	√	√	√	√
hotel reservation	predefined services	alternative	√	√			√	√		
	multiple services				√	√			√	√
flight reservation	predefined services	alternative	√			√	√			√
	multiple services			√	√			√	√	
leisure tour reservation		optional ↰					√	√	√	√
cancel part reservation		required					√	√	√	√

variability management approaches to identify and design services targeted to multiple SOC systems.

In [25], feature analysis and service analysis are combined into a method to guide developers of service-oriented product lines. While the method 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.

In [10], 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 realise a service-oriented systems line in this way. This provides a method to adapt a service-oriented application to different customer needs in changeable environments. The service-oriented systems line consists of two main phases using the business process lines concept, that allows realising 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 [21], 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, i.e., a product of the product line. An application's requirements 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.

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 that incorporates the main principles from both SPLE and SOC, as well as analysis and verification techniques based upon it;
2) To address run-time adaptability through an extension of the scope of the flexibility that can be achieved by introducing variability in service definitions.

In the next section we will outline the research agenda we set up to reach these objectives, as well as elaborate on some specific research that we have already initiated.

III. Towards Variability and Rigour in Service Computing Engineering

The primary objective of the research we envision is to add variability management to the principles of SOC. The overall strategy is to first copy from SPLE mechanisms to include variability notions in software artifacts. 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 in a service-oriented description language, suitably extended to allow for variability management. 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, extending along the same lines some formal verification techniques already available for such languages, should provide the possibility to conduct formal verification over service-oriented specifications that include variability. Variability should be investigated from three points of view:

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

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

A. Modelling Variability and Adaptability

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

Our 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. Once the modelling of variability in SOC is consolidated, the reference model can be extended to address run-time adaptability. However, this is a challenging task that can only be tackled when the modelling of variability in SOC is fully understood (cf. Sect. III-C).

B. Extending Service-Oriented Languages

The idea here is to investigate the extensions and/or modifications that need to be applied to current service-oriented languages to support both static and dynamic variability. This requires to revisit choreography and orchestration and the like. We could start from the mathematically well-founded modelling language SRML and the formal techniques for qualitative and quantitative analysis supporting this approach [1], [16].

SRML offers primitives for modelling *business services* and *activities*, in which interactions are supported through interfaces. It supports a methodological approach that includes using the UMC model checker [9] for qualitative analysis and the Markovian process algebra PEPA [20] for quantitative analysis of timing properties. An advantage of SRML is that modelling is done at a high level of abstraction, where one can assume that the basic mechanisms of SOAs, such as sessions and service discovery/binding, are provided by the middleware and, therefore, need not be part of the models.

We also intend to look for inspiration by studying in detail the formal engineering approach for SOA outlined in [11], since it successfully transfers contributions from the area of formal methods to the application domain of SOA.

Concerning static variability, we need to investigate the most effective ways to include constructs in the languages to express *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), so we expect that a single product respects the syntax of the considered language.

Regarding dynamic variability, several directions can be investigated. 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 called *reconfiguration managers*. A reconfiguration manager is responsible for catching the events indicating the necessity to reconfigure the system, and for reacting by applying the required modifications. 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 proposed enhancements to SOC languages must be validated by experimentation on existing SOC frameworks.

C. Developing Verification Techniques/Tools

By combining and extending the ideas underlying the modelling and analysis and verification techniques and tools that have been developed in the SOC domain with those from the SPLE domain, we aim to develop 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 *reliability*, *availability*, and *responsiveness*. 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 work on the verification of quantitative and qualitative properties on service descriptions can be adapted to be able to deal not only with single services, but directly with service family specifications, thus allowing to factorise both the time and the cost that is needed to verify products that have been correctly derived from a family definition. In this context, we also plan to consider integrating quality models into our formalisation. A starting point might be the specialisation of the quality model for SOA, which is one of the promising outcomes of a German project on the development of software quality standards [14].

A second concern is the use of specific analysis and verification techniques aimed at proving correct derivations of products from service family definitions. Recently, we have used deontic-style logics to model notions of variability in product family descriptions [3]–[8]. Deontic logic [2] provides a natural way to formalise concepts like *violation*, *obligation*, *permission*, and *prohibition* — ideal to express the conformance of members of a product family w.r.t. variability rules. Currently, we are extending the model checker FMC [26] to enable it to check such conformance [7], [8].

In [6], we address the problem of model-checking a family of services, starting from the addition of variability to a simple action-based and branching-time temporal logic interpreted in a deontic way over a basic form of variable transition systems, called modal transition systems. Services are traditionally modelled with richer transition systems, which we intend to address in future work. In particular, FMC is closely related to the CMC model checker for SocL (a service-oriented logic) formulae over the process algebra COWS (Calculus for Orchestration of Web Services) [15]. Adding variability management to CMC will allow us to address families of services that use more complex mechanisms typical of SOC, such as compensation and correlation.

A third concern is the introduction of run-time adaptability, which requires innovative verification techniques and a deep understanding of the relation between sought service properties and variability when defining services. A typical question is whether a SOA satisfies a given property, irrespective of which variant has actually been chosen, or of which adaptation is occurring inside the architecture.

IV. Application Domains

The research activity envisioned in this paper will produce innovative elements to be used to define an engineering methodology for the systematic, large-scale production of *software services* and their market segmentation. These innovative elements shall 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 family's definition through adaptation to client needs or to modification of the specific context for which the application has been deployed. Completion of our research agenda outlined in this paper would thus result in a full-fledged service computing engineering platform — based on rigorous modelling, analysis, and verification techniques — which developers could use to realise applications that use the full potential of SOC.

A challenging technical problem that will need to be tackled for successfully introducing variability modelling in SOC concerns the relation between static and dynamic variability. Incorporating a service within a product line development process is static, while adapting service behaviour at run time is dynamic. Propagation of static constraints to run time may influence the analysis phase, in the sense that it needs to be guaranteed that run-time variability does not break design constraints. Continuing our example from the Introduction, in a product line for travel agencies, the imaginary product "travel agency service specialised in children" should not offer (not even at run time) a feature (service) allowing customers the choice of booking a leisure tour of which it is known that it may be dangerous for children.

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 new ideas proposed in this paper can achieve significant advantages in terms of development costs, as it will be able to factorise common elements as well as the verification processes.

References

[1] J. Abreu, F. Mazzanti, J.L. Fiadeiro, S. Gnesi, A model-checking approach for service component architectures. In *Formal Techniques for Distributed Systems — Proceedings 11th international conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'09)* (D. Lee, A. Lopes, A. Poetzsch–Heffter, eds.), Lecture Notes in Computer Science 5522, Springer, 2009, 219–224.

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

[3] P. Asirelli, M.H. ter Beek, A. Fantechi, S. Gnesi, Deontic logics for modeling behavioural variability. In *Proceedings 3rd international workshop on Variability Modelling of Software-intensive Systems (VaMoS'09)* (D. Benavides, A. Metzger, U. Eisenecker, eds.), ICB Research Report 29, Universität Duisburg–Essen, 2009, 71–76.

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

[5] P. Asirelli, M.H. ter Beek, A. Fantechi, S. Gnesi, A logical framework to deal with variability. In *Proceedings 8th international conference on Integrated Formal Methods (IFM'10)* (D. Méry, S. Merz, eds.), Lecture Notes in Computer Science 6396, Springer, 2010, 43–58.

[6] P. Asirelli, M.H. ter Beek, A. Fantechi, S. Gnesi, A model-checking tool for families of services. In *Formal Techniques for Distributed Systems — Proceedings joint 13th IFIP WG 6.1 international conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'11) and 31st IFIP WG 6.1 international conference on FORmal TEchniques for networked and distributed systems (FORTE'11)* (R. Bruni, J. Dingel, eds.), Lecture Notes in Computer Science 6722, Springer, 2011, 44–58.

[7] P. Asirelli, M.H. ter Beek, A. Fantechi, S. Gnesi, Formal description of variability in product families. In *Proceedings 15th international Software Product Line Conference (SPLC'11)* (E. Almeida, T. Kishi, Ch. Schwanninger, I. John, K. Schmid, eds.), IEEE, 2011, 130–139.

[8] P. Asirelli, M.H. ter Beek, A. Fantechi, S. Gnesi, F. Mazzanti, Design and validation of variability in product lines. In *Proceedings 2nd international workshop on Product LinE Approaches in Software Engineering (PLEASE'11), Companion Proceedings 33nd International Conference on Software Engineering (ICSE'11)*, ACM, 2011, 25–30.

[9] M.H. ter Beek, A. Fantechi, S. Gnesi, F. Mazzanti, A state/event-based model-checking approach for the analysis of abstract system properties. *Science of Computer Programming* 76, 2 (2011), 119–135.

[10] N. Boffoli, M. Cimitile, F. Maria Maggi, G. Visaggio, Managing SOA system variation through business process lines and process oriented development. In [24].

[11] M. Broy, Ch. Leuxner, D. Méndez Fernández, L. Heinemann, B. Spanfelner, W. Mai, R. Schlör, Towards a Formal Engineering Approach for SOA. Technical Report TUM-I1024, Technische Universität München, 2010.

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

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

[14] Consortium project QuaMoCo, http://www.quamoco.de/

[15] A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese, F. Tiezzi, A logical verification methodology for service-oriented computing. To appear in *ACM Transactions on Software Engineering and Methodology*, 2011.

[16] J.L. Fiadeiro, A. Lopes, L. Bocchi, J. Abreu, The SENSORIA reference modelling language. In *Rigorous Software Engineering for Service-Oriented Systems — Results of the* SENSORIA *Project on Software Engineering for Service-Oriented Computing* (M. Wirsing, M. Hölzl, eds.), Lecture Notes in Computer Science 6582, Springer, 2011, 61–114.

[17] B. Geppert, K. Pohl (eds.), *Proceedings 12th international Software Product Lines Conference (SPLC'08)*, IEEE, 2008.

[18] A. Helferich, G. Herzwurm, S. Jesse, Software product lines and service-oriented architecture: A systematic comparison of two concepts. In [13].

[19] A. Helferich, G. Herzwurm, S. Jesse, M. Mikusz, Software product lines, service-oriented architecture and frameworks: Worlds apart or ideal partners? In *Proceedings 2nd international conference on Trends in Enterprise Application Architecture (TEAA'06)* (D. Draheim, G. Weber, eds.), Lecture Notes in Computer Science 4473, Springer, 2007, 187–201.

[20] J. Hillston, *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[21] P. Istoan, G. Nain, G. Perrouin, J.-M. Jézéquel, Dynamic software product lines for service-based systems. In *Proceedings 9th IEEE international conference on Computer and Information Technology*, IEEE, 2009, 193–198.

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

[23] R.W. Krut, S.G. Cohen (eds.), *Proceedings 2nd workshop on Service-Oriented Architectures and Software Product Lines: Putting Both Together (SOAPL'08)*. In [17].

[24] R.W. Krut, S.G. Cohen (eds.), *Proceedings 3rd workshop on Service-Oriented Architectures and Software Product Lines: Enhancing Variation (SOAPL'09)*. In *Proceedings 13th international Software Product Lines Conference (SPLC'09)*, ACM, 2009.

[25] J. Lee, D. Muthig, M. Naab, An approach for developing service oriented product lines. In [17].

[26] F. Mazzanti, FMC v5.0, http://fmt.isti.cnr.it/fmc (2011)

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

[28] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, Service-oriented computing: State of the art and research challenges. *IEEE Computer* 40, 11 (2007), 38–45.

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

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

[31] Ch. Wienands, Studying the common problems with service-oriented architecture and software product lines. Presented at *4th Service Oriented Architecture (SOA) & Web Services conference*, 2006.