# Variability Middleware for Multi-tenant SaaS Applications

## A Research Roadmap for Service Lines

Dimitri Van Landuyt, Stefan Walraven, Wouter Joosen
iMinds-DistriNet, KU Leuven
3001 Leuven, Belgium
{firstname.lastname}@cs.kuleuven.be

## ABSTRACT

Software product line engineering (SPLE) and variability enforcement techniques have been applied to run-time adaptive systems for quite some years, also in the context of multi-tenant Software-as-a-Service (SaaS) applications. The focus has been mainly on (1) the pre-deployment phases of the development life cycle and (2) fine-grained (tenant-level), run-time activation of specific variants. However, with upcoming trends such as DevOps and continuous delivery and deployment, operational aspects become increasingly important.

In this paper, we present our integrated vision on the positive interplay between SPLE and adaptive middleware for multi-tenant SaaS applications, focusing on the operational aspects of running and maintaining a successful SaaS offering. This vision, called Service Lines, is based on and motivated by our experience and frequent interactions with a number of Belgian SaaS providers.

We concretely highlight and motivate a number of operational use cases that require advanced variability support in middleware and have promising added value for the economic feasibility of SaaS offerings. In addition, we provide a gap analysis of what is currently lacking from the perspectives of variability modeling and management techniques and middleware support, and as such sketch a concrete roadmap for continued research in this area.

## Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; D.2.11 [**Software Engineering**]: Software Architectures; D.2.13 [**Software Engineering**]: Reusable Software

## Keywords

Multi-tenant SaaS, Run-time variability, Operational support, Variability middleware, Service Lines, Models at run time



*SPLC 2015, July 20 - 24, 2015, Nashville, TN, USA*



ISBN 978-1-4503-3613-0/15/07…$15.00

DOI: http://dx.doi.org/10.1145/2791060.2791080

## 1. INTRODUCTION

In the last years, Software-as-a-Service (SaaS) has emerged as the most prominent and promising cloud computing service model, and recent forecasts indicate the continuation of this trend for the coming years [4]. In this model, specific software applications are outsourced to the SaaS provider who offers these in an on-demand fashion to many customer organizations (tenants). For these customer organizations, the main benefits are related to the increased flexibility, the limited costs (pay-per-use model and no upfront infrastructural investments), and the fact that they can outsource these typically domain-specific enabling applications that are not part of their core business. For the SaaS provider, higher resource utilization increases the return on investment and to further leverage this, *multi-tenancy* [14, 27] is a common architectural tactic employed by SaaS providers in which a single (or a few) application instances are shared by many tenant organizations (or clusters of tenants).

This focus on increased resource sharing typically results in a one-size-fits-all approach, which lacks support for different and varying requirements of different tenants. To address this form of variability, techniques related to dynamic software product lines and run-time tenant-level customization have successfully been applied [21, 26, 30].

However, SaaS offerings are inherently susceptible to many other forms of variability, caused by (i) co-existence of different versions and variants of key components and enabling services, (ii) variations in deployment (e.g. multi-cloud or hybrid cloud setups), (iii) different involved third-party service providers (SaaS federations), and (iv) tenant management strategies employed by the SaaS provider. These different variability options at play are both of a functional and a non-functional nature (e.g. tenant SLAs).

Furthermore, the high-availability requirements associated to SaaS applications dictate development models that are more continuous and fine-grained ("*develop once/evolve forever*") than the classical discrete software evolution models. This causes a shift of much of the development complexity to the run time and for this, advanced operational support is required.

This paper provides our vision on *Service Lines*, i.e. multi-tenant SaaS applications that are built for run-time customization and for additional operational support for the sake of exploitation (e.g. revenue optimization at the level of individual features), evolution (e.g. upgrading or extending the SaaS offering) and optimization (e.g. cost-effective deployment and migration). Key to this vision is (i) the

use of variability modeling and management techniques to communicate about and enact customization options at runtime, and (ii) sophisticated SaaS middleware that manages this variability complexity by dynamically creating different views on these customization options for different stakeholders (in the form of dedicated SaaS management or configuration dashboards), and by providing these stakeholders with appropriate controls to enact these customizations at run time, in a fine-grained, per-tenant and sometimes staged (i.e. involving multiple stakeholders) manner.

The vision presented in this paper is based on frequent and intensive interactions with Belgian SaaS providers, in the context of a number of collaborative research projects [5, 6, 7, 9]. These SaaS providers are active in widely different domains, ranging from document processing services, to log management and data-intensive security analysis services, to medical workflows and medical imaging services, to engineering process simulation services. The foundations of this paper are rooted firmly on the common needs of these SaaS providers.

This paper is structured as follows. Section 2 presents a number of illustrative and motivational scenarios. Section 3 presents our vision of Service Lines and discusses its underpinning principles. In Section 4, we present our analysis of what is currently lacking to accomplish this vision and as such highlight a number of open research challenges. Finally, Section 5 concludes the paper.

## 2. MOTIVATION

In this section, we present and discuss operational use cases that highlight the potential benefits of systematic middleware support for the different forms of variability affecting multi-tenant SaaS applications.

**Scenario #1: Fine-grained optimization of the SaaS offering.** Instead of relying on static information such as the amount of tenants that indicated interest in a certain feature of the SaaS application, storing usage statistics on a tenant- and feature-level allows creating accurate views (in the form of dashboards) for the SaaS provider that indicate which features are most popular and therefore bring in most revenue. This enables him to make informed strategic decisions, e.g. to improve marketing strategies, to decide which features to remove from the offering altogether, or which features to update and improve (and thus initiating Scenarios #2 and #3), etc. It also allows him to set pricing on a feature basis and dynamically adapt pricing schemes to current demand and supply conditions.

**Scenario #2: Continuous evolution.** Once successful, a SaaS application is a high-availability application, meaning that the SLA violation and service disruption caused by updating and upgrading the SaaS application as a whole (for the sake of maintenance or evolution) in practice becomes unacceptable. This reality increases the need for powerful run-time adaptation mechanisms that allow the SaaS provider to update and upgrade the SaaS offering in a gradual manner, on a fine-grained, per-tenant basis. This scheme of continuous evolution causes the multi-tenant SaaS application to be permanently in an intermediate state of update in which different versions of key components co-exist, and in which different variants are used by different tenants simultaneously.

This introduces a set of operational use cases for (i) keeping track of the different co-existing versions (and keeping their dependencies satisfied), (ii) testing newer versions before activation, (iii) gradually rolling out new versions over clusters of tenants (in accordance with individual tenant SLAs or business requirements), and (iv) eventually phasing out older versions.

**Scenario #3: Continuous delivery.** In addition, the above-mentioned use cases are key to decrease the time to market of new SaaS features. Indeed, the classical development life cycle simply becomes too slow. In true DevOps style [10], delivering new features in multi-tenant SaaS applications is accomplished by employing a streamlined pipeline, covering the phases of (i) inception, (ii) rapid prototyping and small-scale testing, (iii) deployment, (iv) live testing (e.g. A/B testing, or online regression testing), (v) gradual activation over test clusters of tenants, and (vi) activation into production, all within shortened time frames.

The life cycle of a single feature also involves activities such as (i) continuous evaluation based on usage and popularity statistics (e.g. revenue versus operational costs of a feature), and (ii) updating, upgrading and/or gradually phasing out of the feature.

**Scenario #4: SaaS brokerage.** Inter-organizational SaaS federations [3] are increasingly common, and SaaS applications are increasingly built as compositions of other SaaS applications.

One example is the document processing SaaS application in which the final step of the processing workflow involves contacting print and postal services to print and deliver the created documents to their recipients. To accomplish this, the document processing SaaS provider in practice teams up with a number of print and postal SaaS providers, that each have different characteristics: from static (such as the different geographical regions in which they are active), to dynamic (such as the day price for print and delivery), and from functional characteristics (capabilities) to non-functional, SLA-related ones (such as throughput or privacy guarantees).

In such a scheme, the SaaS provider essentially becomes a broker, and is responsible for matching the tenant requirements to a suitable third-party SaaS provider.

**Scenario #5: Deployment flexibility.** Historically, the high cost of private cloud setups (provisioning servers, installing and running the infrastructure) limited SaaS providers. To address this, SaaS offerings are increasingly being delivered on top of public IaaS or PaaS platforms [28], and even on combinations of private and public clouds [25, 29] (so-called hybrid, cross- and multi-cloud deployments).

In such situations, the SaaS provider himself becomes a customer of IaaS or PaaS providers and the need arises to deploy the SaaS application more flexibly (e.g. to avoid vendor, provider or technology lock-in), and to dynamically migrate (parts of) the application to different providers (e.g. when the price drops). For example, in the document processing application, peak demand —typically at the end of the month when many documents such as pay slips and bank statements need to be prepared— is accommodated by dynamically spilling over some processing jobs to external public clouds.

Dealing with this deployment complexity is non-trivial, and selecting the desired deployment configuration is a choice left mostly to the SaaS operator (e.g. based on in-

ternal characteristics such as the planned workload on the private cloud infrastructure), but in some cases involves tenant decisions as well (for example, tenant policies that prohibit patient medical data to leave the continent for privacy reasons).

**Scenario #6: On-demand self-service provisioning.** This principle is about maximally allowing tenants themselves to manage their own service subscription without interacting with the service provider [19], and is key to cloud computing for ensuring the economic feasibility of the SaaS offering. To accomplish this, the SaaS provider typically provides dedicated management dashboard interfaces to the tenant administrator, allowing the tenant to configure the SaaS offering to accommodate their needs. In SaaS offerings, these dashboards are highly application-specific. For example, in the document processing SaaS application, inspection and billing tools utilize application-specific metrics such as "*documents processed per minute*" instead of lower-level metrics such as the amount of CPU hours spent or the used bandwidth. As they are tightly coupled to the SaaS application, building and maintaining such dashboards manually is costly and time-consuming in practice.

Especially in the context of multiple co-existing versions of components, providing individual tenants with customized dashboards that reflect the current state of deployment, yet provide a consistent and workable set of options, in line with tenant SLAs and other tenant properties (e.g. the tenant cluster to which the organization belongs from point of view of the SaaS operator) represents a set of challenging operational use cases.

# 3. SERVICE LINES: VARIABILITY MODELS AT RUN-TIME

Our vision on customizable SaaS applications is called Service Lines and is based on the realization that SaaS providers (beyond dealing with slight variations in tenant requirements) have to manage other forms of variability, not only driven by the tenant organizations, but also the SaaS developers, SaaS operators, SaaS business owners, and external factors such as market circumstances (third-party suppliers entering or leaving the picture, changing alliances and federations, market opportunities, etc).

As such, Service Lines must deal with different forms of variability caused by (i) different tenant requirements, both functional and non-functional (i.e. SLAs), (ii) different tenant management strategies (clustering of tenants), (iii) continuous evolution and gradual adaptation, co-existing versions and variants, (iv) deployment variability, and (v) third-party SaaS integration, dynamically changing federations. These forms of variability represent possible customizations of the service line that are enacted at run time, i.e. as part of the day-to-day operation, exploitation and evolution of the service line, by the stakeholders listed above.

In addition, these forms of variability strongly interact with each other and choices made by one stakeholder may strongly affect the options offered to other stakeholders. Some examples: if the SaaS provider has deployed parts of the application on a high-performance cluster, he can offer this as a premium service to some of his tenants (via SLA options). Another example, variability options offered by a third-party provider might be offered directly to tenants, or early adopter tenants might be offered bleeding edge features, while these may not yet be visible to other tenants before they are well-tested.

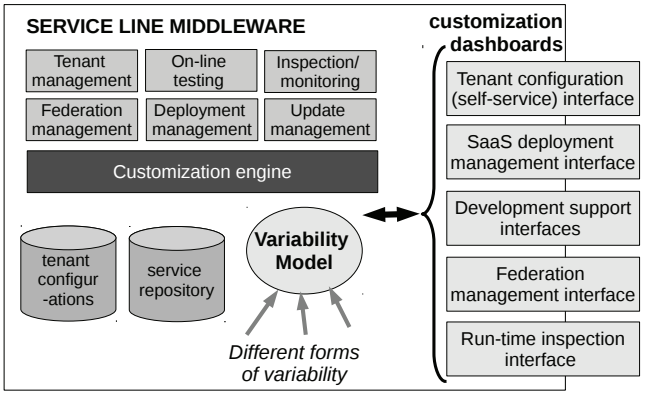


**Figure 1: Schematic overview of the envisioned service line middleware.**

Figure 1 presents a graphical representation of Service Lines, which have the following key characteristics:

**1. Central role of the variability model.** To offer flexible, dynamic and thus successful SaaS offerings, managing the variability complexity outlined above is key. As depicted in Figure 1, the variability model is central to the service line, as it centralizes the different customization options that can be enacted by different stakeholders (tenant, SaaS operators, etc). As opposed to the role of variability models in classical SPL engineering processes [24], we envision the variability model to be a *dynamic, run-time entity* that reflects the current deployment state of the application accurately, and ideally is based upon that (in other words, it is an inspection model built with reflection techniques).

**2. Sophisticated customization middleware.** In addition, our vision on service line postulates sophisticated middleware support for efficiently customizing the SaaS offering *on a per-tenant, per-feature basis*. Figure 1 represents some of these middleware facilities, which rely extensively on the variability model for managing, enacting change, updating the variability model, changing tenant configurations, or the SaaS application itself, etc. On top of these sophisticated middleware facilities, different dashboards can be built for different stakeholders, of which some examples are shown on the right-hand side of Figure 1.

# 4. GAP ANALYSIS

The Service Line vision sketched above may be compelling but introduces many open research challenges. In this section we discuss existing building blocks and analyze the remaining gaps by defining explicit challenges.

## 4.1 Variability modeling and management

Variability modeling and management is a cornerstone of Software Product Line Engineering (SPLE) approaches, in which variability models are commonly used as static, design-time prescriptive models[1], whereas —as mentioned

[1] Even in the context of dynamic software product lines [15, 24], reconfiguration is done at run time, but the variability model itself is created at design time.

earlier— variability models in Service Lines are run-time, flexible entities that are descriptive (i.e. inspection) models of the deployed SaaS application. This idea is in line with the research on models at run-time for adaptive systems [23, 2], and initial ideas on live variability model reconstruction have been proposed [8].

*Challenge V1.* **Dynamic (re-)construction of the variability model**, based on the available variability options (co-existing variants/versions, deployment options, etc).

In literature, distinction is commonly made between different forms of variability [20], e.g. *internal* versus *external* [24], *realization-driven* versus *customer-driven* [21], etc. As discussed in this paper, a multi-tenant SaaS offering is influenced by a number of fundamentally different forms of variability, and variability modeling techniques should provide constructs that distinguish between different sources of variability and link these to the different relevant stakeholders. A first exploration towards extending variability management techniques specifically for Service Lines has been done in earlier work [13].

*Challenge V2.* **Multi-view variability modeling techniques**, expressing different forms of variability, different views on the variability model for different stakeholders.

Managing, expressing and exploiting the different interactions between variability options is additionally challenging, and there is a substantial risk of introducing fragility due to the centralized nature of the variability model. There is a lack of research into *decentralized* feature models, i.e. feature models that are dynamically constructed as a composition of the meta-information of the deployed and activated features.

*Challenge V3.* **Decentralized (re-)construction of the variability model**

## 4.2 Service Line Middleware

Much related work exists about how to architect, design, and structure a service line [18, 21, 26, 27]. In earlier work, we have defined a method for engineering Service Lines [30] to support variability driven by different tenant requirements.

However, as outlined in this paper, classical development methodologies are being challenged by more agile DevOps-inspired approaches of gradual and continuous integration, delivery, evolution and improvement [17, 16].

More specifically, middleware support is required for (i) on-line testing (e.g. change impact analysis, A/B testing, or regression testing based on live or recent data), (ii) phased delivery: activating features gradually, at the basis of individual tenants (or clusters) (e.g. early adopter tenants or strategic partners get updates first), and (iii) staged reconfiguration [12, 26], i.e. reconfiguration actions that require coordination between multiple stakeholders and are tailored to the nature of the adaptation (e.g. tenant administrators must agree to the time at which an update is planned).

*Challenge MW1.* **Integrated support for on-line testing**
*Challenge MW2.* **Support for phased/gradual delivery**, e.g. over tenant clusters.
*Challenge MW3.* **Support for staged reconfiguration**, involving different stakeholders.

Much of these use cases are already state of practice for big SaaS companies such as Amazon and Google [1, 16], but to our knowledge, no reusable SaaS middleware exists with systematic built-in support.

Additional challenges are related to dealing with the complexity introduced by the desired deployment flexibility (cf. *Challenges V1 and V3*), i.e. multi-, cross- or hybrid cloud deployment [29], dynamic migration [11, 25, 29], feature placement [22].

*Challenge MW4.* **Configuration interfaces that dynamically adapt to the current deployment situation**
*Challenge MW5.* **Support for hybrid, cross- and multi-cloud deployments**
*Challenge MW6.* **Tenant-level policy-driven reconfiguration**, e.g. the dynamic spill-over scenario.

Little related work exists on dealing with the complexity of SaaS federations:

*Challenge MW7.* **Federation management facilities**, integration of 3rd party SaaS services/brokerage/reselling.
*Challenge MW8.* **Configuration interfaces that dynamically adapt to the current federation**, (cf. *Challenges V1&V3*).

And likewise, challenges related to optimizing the SaaS offering are to our knowledge generally unaddressed.

*Challenge MW9.* **Run-time fine-grained architecture inspection**
*Challenge MW10.* **Gathering fine-grained application-specific SaaS metrics**, e.g. usage statistics (popularity, revenue, resource utilization) optimizating the SaaS offering.

Addressing these challenges requires advanced middleware support allowing the SaaS offering to be inspected, operated, exploited, and optimized in terms of features (i.e. *feature middleware* in support of a feature-oriented dominant decomposition). Central to this is the variability model as an accurate and detailed representation of the current offering (cf. *Challenges V1, V2 and V3*).

# 5. CONCLUSION

This paper presents Service Lines, our vision on the benefits of applying variability management techniques and adaptive middleware in support of multi-tenant Software-as-a-Service (SaaS) applications. We provide some illustrative examples of the potential benefits that can be gained from more systematic support of variability in SaaS middleware, especially during the operation and exploitation of the SaaS application. In addition, we present our gap analysis of existing work and identified promising open research challenges.

The use of SPL-related techniques to support variability in SaaS applications is not new and a number of point solutions have been defined. However, to our knowledge, no integrated research vision has been mapped out specifically for multi-tenant SaaS applications, and the use of variability models not in a prescriptive but in a descriptive manner is compelling yet highly challenging.

**Acknowledgments.** This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund KU Leuven (project GOA/14/003 - ADDIS). The companies and organizations involved in the realization of this work are Televic Healthcare, Luciad, UP-nxt, Verizon Terremark, Noesis and Agfa Healthcare.

## 6. REFERENCES


[1] S. Ajmani, B. Liskov, and L. Shrira. Modular software upgrades for distributed systems. In *ECOOP*. Springer, 2006.

[2] G. H. Alférez and V. Pelechano. Dynamic evolution of context-aware systems with models at runtime. In R. B. France, J. Kazmeier, R. Breu, and C. Atkinson, editors, *Model Driven Engineering Languages and Systems*, volume 7590 of *LNCS*, pages 70–86. Springer Berlin Heidelberg, 2012.

[3] N. M. Calcavecchia, A. Celesti, and E. Di Nitto. Understanding decentralized and dynamic brokerage in federated cloud environments. *Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice: Theory and Practice*, page 36, 2012.

[4] L. Columbus. Roundup of cloud computing forecasts and market estimates. http://www.forbes.com/sites/louiscolumbus/2015/01/24/roundup-of-cloud-computing-forecasts-and-market-estimates-2015/, January 2015. Last visited on April 9, 2015.

[5] CUSTOMSS. CUSTOMization of Software Services in the cloud (iMinds ICON project), 2011.

[6] D-Base. Decentralized support for Business processes in Application Services (iMinds ICON project), 2014.

[7] DeCoMAdS. Deployment and Configuration Middleware for Adaptive Software-as-a-Service (SBO project), 2014-2018.

[8] M. Denker, J. Ressia, O. Greevy, and O. Nierstrasz. Modeling features at runtime. In *Model Driven Engineering Languages and Systems*, pages 138–152. Springer, 2010.

[9] [DMS]2. Decentralized Data Management and Migration for SaaS (iMinds ICON project), 2014.

[10] B. Fitzgerald and K.-J. Stol. Continuous software engineering and beyond: Trends and challenges. In *1st International Workshop on Rapid Continuous Software Engineering*, pages 1–9. ACM, 2014.

[11] Y. Fu, Y. Wang, and E. Biersack. A general scalable and accurate decentralized level monitoring method for large-scale dynamic service provision in hybrid clouds. *Future Generation Computer Systems*, 29(5):1235–1253, 2013.

[12] F. Gey, D. Van Landuyt, W. Joosen, and V. Jonckers. Continuous evolution of multi-tenant SaaS applications: A customizable dynamic adaptation approach. In *PESOS*, May 2015.

[13] F. Gey, D. Van Landuyt, S. Walraven, and W. Joosen. Feature models at run time: Feature middleware for multi-tenant SaaS applications. In *Models@run.time '14*, pages 21–30. CEUR-WS.org, 2014.

[14] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao. A framework for native multi-tenancy application development and management. In *CEC/EEE '07: 9th IEEE International Conference on E-Commerce Technology and 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*, pages 551–558, 2007.

[15] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic software product lines. In *Systems and Software Variability Management*, pages 253–260. Springer, 2013.

[16] J. Humble. The case for continuous delivery. http://www.thoughtworks.com/insights/blog/case-continuous-delivery, February 2014.

[17] J. Humble and D. Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[18] R. Krebs, C. Momm, and S. Kounev. Architectural concerns in multi-tenant SaaS applications. In *CLOSER '12: 2nd International Conference on Cloud Computing and Services Science*, 2012.

[19] P. Mell and T. Grance. The NIST definition of cloud computing. Special Publication 800-145, National Institute of Standards and Technology (NIST), 2011.

[20] A. Metzger, K. Pohl, P. Heymans, P. Schobbens, and G. Saval. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*, pages 243–253. IEEE, 2007.

[21] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl. Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications. In *PESOS '09: ICSE Workshop on Principles of Engineering Service Oriented Systems*, pages 18–25. IEEE Computer Society, 2009.

[22] H. Moens and F. De Turck. Feature-based application development and management of multi-tenant applications in clouds. In *SPLC '14: 18th International Software Product Line Conference*, pages 72–81, 2014.

[23] B. Morin, O. Barais, J. Jezequel, F. Fleurey, and A. Solberg. Models@run.time to support dynamic adaptation. *Computer*, 42(10):44–51, 2009.

[24] K. Pohl, G. Böckle, and F. Van Der Linden. *Software product line engineering: Foundations, principles, and techniques*. Springer-Verlag New York Inc., 2005.

[25] A. Rafique, S. Walraven, B. Lagaisse, T. Desair, and W. Joosen. Towards portability and interoperability support in middleware for hybrid clouds. In *CrossCloud '14: 1st IEEE INFOCOM CrossCloud Workshop*, pages 7–12. IEEE, 2014.

[26] J. Schroeter, P. Mucha, M. Muth, K. Jugel, and M. Lochau. Dynamic configuration management of cloud-based applications. In *SPLC '12: 16th International Software Product Line Conference - Volume 2*, pages 171–178. ACM, 2012.

[27] S. Walraven, E. Truyen, and W. Joosen. A middleware layer for flexible and cost-efficient multi-tenant applications. In *Middleware '11: 12th International Conference on Middleware*, pages 370–389. Springer Berlin / Heidelberg, 2011.

[28] S. Walraven, E. Truyen, and W. Joosen. Comparing PaaS offerings in light of SaaS development. *Computing*, 96(8):669–724, 2014.

[29] S. Walraven, D. Van Landuyt, A. Rafique, B. Lagaisse, and W. Joosen. PaaSHopper: Policy-driven middleware for multi-PaaS environments. *Journal of Internet Services and Applications*, 6(1), 2015.

[30] S. Walraven, D. Van Landuyt, E. Truyen, K. Handekyn, and W. Joosen. Efficient customization of multi-tenant Software-as-a-Service applications with service lines. *Journal of Systems and Software*, 91:48–62, 2014.