User-Centric Adaptation Analysis of Multi-Tenant Services

JESÚS GARCÍA-GALÁN, University of Seville
LILIANA PASQUALE, Lero - the Irish Software Research Centre
PABLO TRINIDAD and ANTONIO RUIZ-CORTÉS, University of Seville

Multi-tenancy is a key pillar of cloud services. It allows different users to share computing and virtual resources transparently, meanwhile guaranteeing substantial cost savings. Due to the tradeoff between scalability and customization, one of the major drawbacks of multi-tenancy is limited configurability. Since users may often have conflicting configuration preferences, offering the best user experience is an open challenge for service providers. In addition, the users, their preferences, and the operational environment may change during the service operation, thus jeopardizing the satisfaction of user preferences. In this article, we present an approach to support user-centric adaptation of multi-tenant services. We describe how to engineer the activities of the Monitoring, Analysis, Planning, Execution (MAPE) loop to support user-centric adaptation, and we focus on adaptation analysis. Our analysis computes a service configuration that optimizes user satisfaction, complies with infrastructural constraints, and minimizes reconfiguration obtrusiveness when user- or service-related changes take place. To support our analysis, we model multi-tenant services and user preferences by using feature and preference models, respectively. We illustrate our approach by utilizing different cases of virtual desktops. Our results demonstrate the effectiveness of the analysis in improving user preferences satisfaction in negligible time.

Categories and Subject Descriptors: H.1.2 [**Information Systems Applications**]: User/Machine Systems—*Human information processing*; H.4.2 [**Information Systems Applications**]: Types of Systems—*Decision support*

General Terms: Algorithms, Human Factors

Additional Key Words and Phrases: User systems, human information processing, multi-tenant services

ACM Reference Format:
Jesús García-Galán, Liliana Pasquale, Pablo Trinidad, and Antonio Ruiz-Cortés. 2016. User-centric adaptation analysis of multi-tenant services. ACM Trans. Auton. Adapt. Syst. 10, 4, Article 24 (January 2016), 26 pages.
DOI: http://dx.doi.org/10.1145/2790303

1. INTRODUCTION

Multi-tenancy allows cloud providers to deliver the same service to different customers who share physical and/or virtual resources transparently [Bezemer et al. 2010; Natis 2012]. Depending on the adopted cloud service model, users can share resources at

This work was partially supported by the European Commission (FEDER), the Spanish and the Andalusian R&D programmes (grants TIN2012-32273 (TAPAS), P12-TIC-1867 (COPAS) and TIC-5906 (THEOS)), the Science Foundation Ireland grants 10/CE/I1855 and 13/RC/2094, and the ERC Advanced Grant (ASAP) no. 291652.
Authors' addresses: J. García-Galán, P. Trinidad, and A. Ruiz-Cortés, Dpto. Lenguajes y Sistemas Informáticos. ETS Ingeniería Informática, Avda Reina Mercedes s/n. 41012, Seville, Spain; emails: {jegalan, ptrinidad, aruiz}@us.es; L. Pasquale, Lero – the Irish Software Research Centre, Tierney Building, University of Limerick, Ireland; email: liliana.pasquale@lero.ie.

DOI: http://dx.doi.org/10.1145/2790303

different levels, from hardware resources (e.g., CPU, storage) to software applications. Multi-tenancy can support different degrees of isolation. In particular, the lower the degree of isolation, the bigger the resources and cost savings, but the smaller the configurability. Limited configurability [Mietzner et al. 2009] is a major drawback, especially when user preferences are not known in advance. Several approaches [Baresi et al. 2012; Kumara et al. 2013; Mietzner et al. 2009; Schroeter et al. 2012a, 2012b] have been proposed to support dynamic configuration management of multi-tenant services. Nonetheless, these contributions consider an isolated multi-tenant model (i.e., each user is assigned to a different service instance) and focus on deploying different isolated variants of a service instance at runtime.

Social adaptation [Ali et al. 2012] considers changes in the user collective judgment as a new adaptation driver. Other approaches [Malek et al. 2012; Cardellini et al. 2012] consider conflicting users' preferences and limited infrastructural resources in the construction of adaptive software systems. These approaches identify a system architecture [Malek et al. 2012] or configure a service-oriented application that maximises QoS preferences [Cardellini et al. 2012] when changes in the operational environment take place. However, as far as we are aware, a user-centric approach has not been previously proposed for the adaptation of multi-tenant services in cloud scenarios. To achieve this aim, additional challenges have to be addressed. First, it is necessary to provide users with high-level mechanisms to define and change their preferences on the possible service configurations. Second, the adoption of a pay-as-you-go business model allows users to join and leave a cloud service dynamically, which can have an impact on the consumption of the infrastructural resources and may reduce the satisfaction of the user preferences. Therefore, changes in the number of users and modifications of their preferences have to be considered as a main adaptation trigger for the reconfiguration of multi-tenant services.

In this article, we characterize the user-centric adaptation of multi-tenant services problem, focusing on adaptation analysis. In our previous work [García-Galán et al. 2014], we highlighted the challenges to be addressed for engineering the activities of the Monitoring, Analysis, Planning, Execution (MAPE) loop [Kephart and Chess 2003] necessary to support user-centric adaptation and proposed a preference-based analysis that maximizes in a balanced way the satisfaction of user preferences expressed on the possible service configurations. In this article, we extend our analysis by incorporating infrastructural and obtrusiveness aspects. Infrastructural aspects are necessary to guarantee that available infrastructural resources can handle the workload generated by the selected service configuration and the tenants of the service. Obtrusiveness aspects are taken into account to reduce the nuisance produced by a service reconfiguration. Our adaptation analysis can be triggered when users or the operational environment (including available service configurations and infrastructural resources) change at runtime.

We illustrate and motivate our approach by utilizing different cases of virtualized desktops. Compared to our previous proposal, which focused on a hosted shared *Desktop as a Service (DaaS)*, this article extends the applicability of our analysis to different DaaS delivery models and, more generally, to different multi-tenant services. We model the available service configurations (also referred to as configuration space), the infrastructural resources , and the workload by using *Feature Models (FMs)* [Kang et al. 1990]. We represent user preferences by adopting an existing preference model [García et al. 2013]. Our adaptation analysis is interpreted as a multi-objective constrained optimization problem built on top of the *Automated Analysis of Feature Models (AAFM)* [Benavides et al. 2010]. The optimization problem is solved by using metaheuristic algorithms [Marler and Arora 2004] that have proved suitable for FM optimization in existing work [Guo et al. 2011; Sayyad et al. 2013]. We evaluate the effectiveness of

the analysis on simulated scenarios where different tenants—and their users—join and leave a DaaS and change their preferences. The results obtained from our experimental evaluation are encouraging because they demonstrate that our adaptation analysis is able to calculate reconfigurations that improve and balance the satisfaction of users' preferences in a few seconds.

The rest of the article is organized as follows: Section 2 provides some background on multi-tenancy, DaaS, and feature modeling. Section 3 illustrates our DaaS case study, and Section 4 introduces the user-centric adaptation problem and, in particular, the adaptation analysis. Section 5 describes how multi-tenant services and preferences are modeled to support the analysis, which in turn is presented in Section 6. Section 7 discusses our experimental results, and Section 8 points out the open issues that will be addressed in future work. Section 9 compares our approach with relevant related work, and, finally, Section 10 concludes.

2. BACKGROUND

This section provides some background on multi-tenancy, DaaS delivery models, and feature models.

2.1. Multi-Tenancy

Multi-tenancy is defined as *multiple customers, organizations, or processes (tenants) sharing common physical or virtual computing resources while remaining logically independent* [Natis 2012]. Typically, a tenant *groups a number of users, which are the stakeholders in the organization* [Bezemer et al. 2010]. Shared resources can vary depending on the cloud service model; each model provides resources belonging to different levels of abstraction. The *Infrastructure as a Service (IaaS)* model offers computer—physical or virtual machines—and other resources, such as raw block storage, file or object storage, Virtual Local Area Networks (VLANs), IP addresses, and firewalls. To deploy their applications, users install operating system images and their application software on the cloud infrastructure. In the *Platform as a Service (PaaS)* model, providers deliver a computing platform, typically including an operating system and a solution stack with Database Management Systems (DBMS) and/or application servers. Cloud users can run their software solutions without managing the underlying hardware and software layers. In the *Software as a Service (SaaS)* model, users can access applications and data. The more resources are managed by cloud providers, the more resources are shared by multiple different users.

A *Desktop as a Service (DaaS)* is a specific case of SaaS providing a virtual desktop and a set of applications as a service to a single or multiple tenants. Providers like Citrix[1], VMWare[2], and Amazon[3] are increasingly offering a wide range of DaaS solutions. In this article, we focus on the case of multi-tenant DaaS relying on the delivery models provided by Citrix [2013].

2.2. Desktop as a Service Delivery Models

DaaS delivery models differ depending on the specific provider. In particular, as shown in Figure 1(a), Citrix provides two main delivery models for DaaS: *Hosted Shared* and *Virtual Desktop Infrastructure (VDI)*.

The hosted shared model consists of multiple user desktops shared among different tenants and hosted on a single server-based operating system. Although it provides a low-cost, high-density solution, applications must be compatible with a multi-user

[1]http://www.citrix.com/solutions/desktop-as-a-service/.
[2]http://www.vmware.com/products/desktop-virtualization.
[3]http://aws.amazon.com/workspaces/.

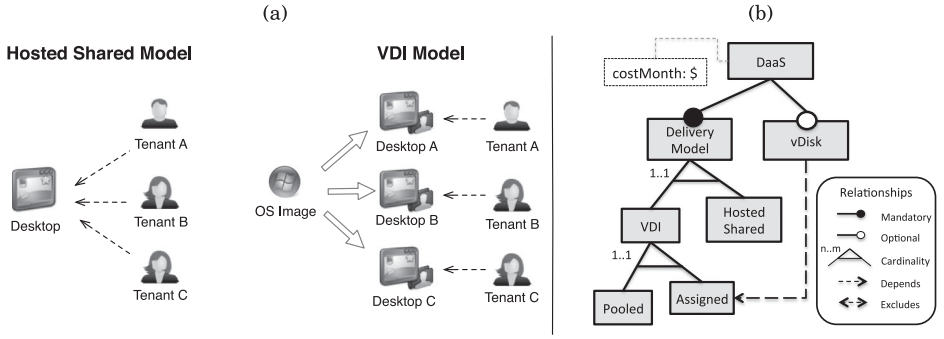

Fig. 1. (a) DaaS delivery models. (b) Example of a FM.

Table I. Impact of User Profiles on Required Infrastructural Resources for Different DaaS Delivery Models [Citrix 2013]

		CPU: users per core (MIPS per user)			RAM per user	
Profile	Apps	Pooled VDI	Assigned VDI	Shared	VDI	Shared
Light	1–2 office apps	15 (1,340)	13 (1,546)	21 (957)	1GB	340MB
Normal	2–10 office apps. light multimedia use	11 (1,827)	10 (2,010)	14 (1,435)	2GB	500MB
Heavy	Multimedia or app development	6 (3,350)	5 (4,020)	7 (2,871)	4GB	1GB

We assume a Windows server 2012 and Windows 8 for VDI and hosted shared models, respectively, and a processor speed of 2.7GHz and Intel Westmere processor architecture.

server-based operating system. In addition, because multiple users are sharing a single operating system, they are prevented from performing actions that may negatively affect other users, such as installing new applications or changing system settings.

The VDI model hosts custom desktop instances on remote servers. Each desktop instance is associated with a different tenant and relies on a centralized master image. VDI supports more customization than the hosted shared model since each tenant uses a different desktop instance. However, the specific shared and exclusive aspects depend on the concrete VDI implementation and its options. For example, Citrix supports *pooled VDI* and *assigned VDI* (with personal vDisk) [Citrix 2013]. A pooled VDI provides a clean random virtual desktop each time a user accesses the service. An assigned VDI allows the users to customize the desktop, save applied changes after logging out, and connect to the same virtual machine at each login.

The choice of a delivery model depends on the number of users, their profiles (i.e., intensity of desktop usage) and diversity, the applications adopted more often, and the available infrastructural resources. In the case of a hosted shared model, all the features are shared among all the tenants, whereas in the case of a VDI model, the shared aspects depend on the concrete VDI configuration. Table I shows the impact of different user profiles on the required infrastructural resources (CPU and RAM) for different DaaS delivery models. CPU requirements determine the maximum number of users of a specific type that can be allocated for each core, whereas RAM requirements indicate the amount of RAM (in MB or GB) necessary to serve the requests of each user. In the next section, we present a DaaS case study for the aforementioned delivery models.

2.3. Feature Models

Feature Model (FMs) [Kang et al. 1990] are used to represent all the possible products that can be built in variability-intensive systems such as *Software Product Lines (SPLs)*. FMs are tree-like data structures where each node represents a product feature. Features are bound by means of hierarchical (mandatory, optional, and set) and

Table II. Shared DaaS Configuration Options Depending on Delivery Model and Their Workload Peaks

Shared options			DaaS configuration options		Values	Workload Peaks	
HS	PVDI	AVDI				MIPS	RAM
✓	✗	✗	Regional Settings		{UK, US, ES}	-	-
✓	✗	✗	Gadgets	Weather	{On, Off}	-	-
				Calendar		-	-
✓	✓	✗	Maintenance	Defragmenter	{On, Off}	10 000	3 GB
				Indexing		15 000	4 GB
				Backup	{Daily, Weekly, Monthly}	7 000	1 GB
✓	✓	✗	Updates	Java	{Daily, Weekly, Monthly}	3 000	0.2 GB
		✗		Eclipse		2 000	0.1 GB
		✗		MS-Office *		3 000	0.4 GB
		✓		OS *		5 000	1 GB

*MS-Office updates period should be smaller than that used for OS updates.

cross-tree relationships. These relationships define how features can be combined in a product, thus defining the configuration space of the system. Figure 1(b) shows an example of an FM diagram that represents the variability of DaaS delivery models. `DaaS` is the root feature that represents the overall functionality of the system. It has two children, an optional feature (white circle) named `vDisk`, and a mandatory feature (black circle) named `Delivery Model`. The latter feature is decomposed by a set relationship whose cardinality indicates the number of child features that can be chosen at the same time. Note that in an FM, non-leaf features [Thum et al. 2011] can be used to represent high-level decisions and group lower-level decisions. For example, the `VDI` feature is used to group two possible VDI implementations (`Pooled` and `Assigned`).

FMs also represent cross-tree constraints, attributes, and complex constraints. Cross-tree constraints are constraints between features belonging to different branches of the model, such as the dependency between features `vDisk` and `Assigned`, indicating that the adoption of a personal vDisk requires the selection of an assigned VDI implementation. Attributes are additional properties associated with a feature. For example, the `totalCost` attribute of the `DaaS` feature is a real number describing the cost of a specific DaaS configuration. Finally, complex constraints describe arithmetic, logical, and relational constraints on features and attributes. They can be used, for example, to bound the possible values that attributes can assume.

3. CASE STUDY

In this section, we present a Windows-based DaaS case study that we use to motivate and illustrate our work. First, we describe the configuration space of the DaaS that can be delivered by using a hosted shared or VDI model. Second, we illustrate how different user profiles and DaaS configuration options impact on the infrastructure (CPU and RAM). Finally, we show a multi-tenant scenario in which each tenant groups users having compatible DaaS configuration preferences.

3.1. DaaS Configuration Space

We present a Windows-based DaaS example in which each instance uses different delivery models: hosted shared (HS), pooled (PVDI), and assigned VDI (AVDI). We assume that every DaaS instance provides several applications: (1) a LaTex compiler and editor, (2) MS-Office, (3) a PDF reader, (4) GIMP as image editor, (5) Eclipse as IDE, and (6) SPSS for statistical analysis. An instance setup is defined by four configuration options: regional settings, gadgets (desktop widgets), maintenance tasks, and updates frequency. Tenants can indicate their preferred configuration options, and the satisfaction of such preferences depends on the delivery model. Table II indicates the

Table III. Tenants' Usage Profile and Preferences; Potential Conflicting Preferences Are Associated with the Same Number

	# Users	Apps	Profile	Preferences
Tenant 1	45	- MS-Office - GIMP	Medium	- US reg. settings (1) - Office updates (3) - Weekly backups - Weekly OS Updates (3)
Tenant 2	60	- Latex - MS-Office	Light	- No UK regional settings (1) - Indexing (2) - Defragmentation
Tenant 3	31	- Eclipse - MS-Office - PDF Reader	Heavy	- Monthly office updates (3) - No Indexing (2) - Calendar Gadget

configuration options that are shared in the different delivery models. If a configuration option is shared by multiple tenants, conflicts among different preferences may arise. For example, in a hosted shared model, different tenants may have different update frequency preferences for Eclipse and MS-Office, whereas in an assigned VDI model, such an option can be customized for each tenant, thus avoiding any possible conflict.

3.2. Infrastructural Constraints

In this section, we characterize the workload generated by the service users and its configuration in order to assess whether the infrastructural resources available at the service provider are satisfactory to provision a DaaS instance while avoiding service outages. The workload generated by the activity of each user depends on the applications he or she executes more often. We profile users along the three categories identified by Citrix [2013]: light, normal, and heavy. The delivery model has an impact on the number of users per core that a DaaS is able to handle and on the required RAM size. Table I shows an estimation of the workload generated by each user profile depending on the DaaS delivery model and expressed in terms of *Million Instructions Per Second (MIPS)* and memory size.[4] In addition to the workload generated by the users, the current service configuration also has an impact on the workload. Although this impact is not publicly described by providers, it can be easily profiled in a system. For the possible DaaS configuration options envisaged in our example scenario, we assume to have the peak workloads shown in the last two columns of Table II. Note that a peak workload represents the maximum workload that can be reached at a given time instant.

3.3. Users, Preferences, and Conflicts

In our example scenario, we assume three different tenants sharing the same DaaS instance. Each tenant groups a given number of similar users whose preferences, used applications, and profiles are presented in Table III. Preferences expressed on the same configuration options may lead to conflicts. For example, tenants 1 and 2 have different—but equivalent—preferences for the regional settings that do not create conflicts. Tenants 2 and 3 have contradicting preferences for the indexing feature, making it impossible to satisfy both preferences at the same time. Tenant 1 prefers weekly OS updates, while tenant 3 favors monthly office updates, causing a potential conflict.

[4]Whereas the RAM and users per core calculations are provided by Citrix, the MIPS are estimated based on the MIPS of an Intel Westmere Core i7 980X (hex-core) 3.3Ghz processor. We have to adjust the clock frequency to the Westmere 2.7Ghz of Table I. The MIPS for a single core are $\frac{147,600*2.7}{6*3.3} \approx 20100$.

Indeed, the satisfaction of both preferences may cause a violation of the constraint that requires that the office updates period must be smaller than the OS update period (Table II). Note that the complete satisfaction of all preferences is infeasible in most cases and therefore it is necessary to trade off conflicting preferences.

Similarly to other cloud service models, a multi-tenant DaaS satisfies the requests of its tenants elastically. This means that the tenants may join and leave the service or change their preferences or number of users at runtime. For example, the users in tenant 3 work at fixed times and they therefore join and leave the DaaS almost at the same time every workday; however, users belonging to the rest of the tenants access the desktop at different times (including weekends), especially when project deadlines are close. Similarly, the current service configuration may become suboptimal because tenant preferences vary during the system lifetime. For example, users of tenant 3 may prefer to deactivate Eclipse updates while finishing a development sprint. In all these cases, the current users and their preferences have a direct influence on the selection of a specific service configuration. Furthermore, modifications to the available service configurations might lead to changes in tenant preferences. For example, if some of the backup features are removed, the users might change their preferences with regard to the new configuration space.

4. TOWARD USER-CENTRIC ADAPTATION OF MULTI-TENANT SERVICES

In this section, we present the foundations of our user-centric adaptation approach for multi-tenant services. In particular, we provide a big picture of the user-centric adaptation problem and focus on the analysis for supporting service reconfiguration.

4.1. User-Centric Adaptation Problem

We consider the user-centric adaptation as the process that reconfigures a system when the users or the operational environment change in order to maximize user satisfaction. The adaptation actions perform a system reconfiguration by changing the values of the configuration options. We propose to perform the adaptation when any event that may have an impact on user satisfaction is detected, such as changes in the user preferences, in the available system configurations, or in the computational resources. However, system adaptations can, in turn, reduce system usability. Therefore it is necessary to balance the tradeoff between preferences satisfaction and the obtrusiveness of adaptation actions.

As shown in Figure 2, the activities of the MAPE loop can support user-centric adaptation as follows:

(a) **Monitoring (M)** has the objective of capturing user-related changes, modifications of the available system configurations, and variations of computational resources. Any of these changes can trigger a new adaptation. User-related changes include modifications of the users' preferences on the available system configurations or variations in the number of users per tenant, which in turn can affect the global preferences of a tenant. Monitoring users' preferences can be performed, for example, by asking for explicit user feedback [Ali et al. 2012]. Modifications of the configuration space may be due to, for example, new applications supported by the DaaS or system updates. Infrastructure changes are related to modifications of allocated resources or changes of the constraints on the maximum resources that can be allocated. Note that all these changes can have an impact on how users' preferences are satisfied. Additionally, the monitoring also has to keep track of the adaptation frequency, which may affect the performance of tasks performed by users [Speier et al. 2003] and therefore preclude the execution of a reconfiguration.

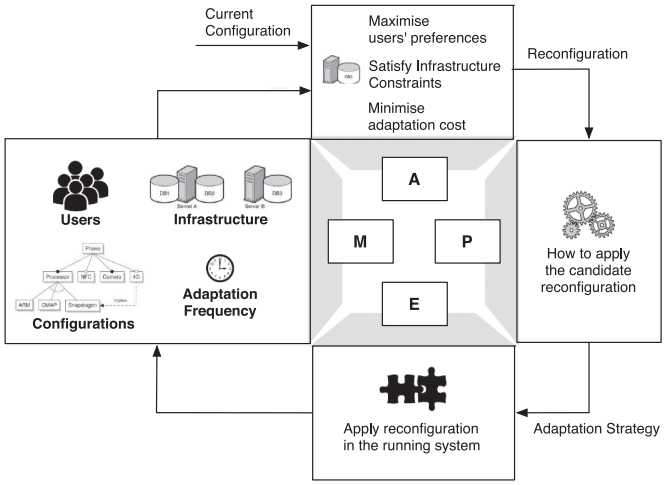

Fig. 2. User-centric adaptation Monitoring, Analysis, Planning, Execution (MAPE) loop.

(b) **Analysis (A)** has the objective of identifying the best system configuration(s) that optimize a set of metrics. In particular, a reconfiguration should maximize the satisfaction of user preferences by taking into account the available infrastructural resources. As changes from one configuration to another can have a negative impact on the usability of the system [Gajos et al. 2006], the reconfiguration should also minimize the adaptation cost (a.k.a "obtrusiveness"). For example, a reconfiguration that modifies the look and feel or the regional settings in a DaaS is more obtrusive than another one that modifies the backup frequency. Frequent adaptations can also increase obtrusiveness. This issue is further discussed and parametrized in Sections 5.1 and 6.3, respectively.

(c) **Planning (P)** receives as input a candidate reconfiguration identified during analysis and identifies an adaptation strategy indicating how this reconfiguration should be applied at runtime. For example, changes in the application look and feel might not be applied until specific users terminate their interaction with the system. A reconfiguration that modifies the backup frequency can only be applied after the next scheduled backup.

(d) **Execution (E)** has the objective of applying an adaptation at runtime. For example, in the case of a VDI DaaS, a variant of existing application instances should be deployed dynamically, as proposed in Baresi et al. [2012] and Schroeter et al. [2012b], whereas for a hosted shared DaaS model, the single application instance should be modified when possible.

For our DaaS case study, depending on the chosen delivery model, the value of a configuration option can be tenant-specific (i.e., enabling a different configuration for each tenant) or tenant-shared (i.e., common to all tenants). However, as a multi-tenant service, all the delivery models present a—higher or lower—number of shared configuration options. We propose to adapt the shared configuration options dynamically. In this way, our approach can be applied to different delivery models by changing the options that are included in the configuration space considered during the analysis. Note that admin aspects that are common to all the tenants, such as security configurations (e.g., firewall, antivirus), are out of the scope of our adaptation problem. Indeed, given their criticality, their configuration can only be performed by the admin staff at the provider organization.

Our user-centric adaptation approach can be applied to other multi-tenant service models, such as SaaS, PaaS, and IaaS. An example of multi-tenant SaaS is Wordpress,[5] which is an open source blogging tool and a content management system providing different customization options and plugins. Similarly to a DaaS, Wordpress supports multi-sites,[6] which aggregate several Wordpress sites into a single installation. In this case, the shared resources among tenants are the global configuration options, such as the default language, the upgrading policy, and the available plugins, themes, and blog entries. In a PaaS model, the deployment environment (e.g., DBMSs, web servers) can be reconfigured to adequately host multiple applications—representing the tenants in this case—having different needs. Virtualized computing instances and storage services are examples of multi-tenant IaaS. Such services rely on the underlying hardware resources that are shared among different tenants. For example, Amazon offers micro instances to increase CPU capacity for a short time in order to handle load peaks. Since micro instances do not have fixed performance requirements, in this scenario our approach can be used to decide which micro instance receives additional computational cycles. This choice depends on the current and changing needs of the tenants and on the available computational capacity of the physical CPU instance shared among the tenants.

4.2. Adaptation Analysis for Service Reconfiguration

In this article, we focus on the analysis activity of the MAPE loop. In particular, we define the analysis problem as a tuple of the form

$$(C, I, T, f_C, f_T, (u_1, \ldots, u_n), \rho),$$

such that C represents the set of configurations that are available in the service; I characterizes the infrastructural resources; T represents the set of tenants, $f_C : C \rightarrow I$ and $f_T : T \rightarrow I$ are the functions that identify the impact (workload) that each configuration and each tenant has on the required infrastructural resources, respectively; $u_i : C \rightarrow \mathbb{R}$ are utility functions for each tenant in T defining the tenant satisfaction for any given configuration in C; and $\rho : C \times C \rightarrow \mathbb{R}$ is a function that quantifies the obtrusiveness that a change from one configuration to another produces.

Assuming that it is possible to define a function $U : C \rightarrow \mathbb{R}$ that computes the global satisfaction of all the tenants for a given configuration, a candidate reconfiguration $c_{t+1} \in C$ can only be enforced if it outperforms the current one (c_t) (i.e., $U(c_{t+1}) \geq U(c_t) + \rho$). Note that we assume that a tenant groups different users who share compatible preferences and the same profile. The clustering of the user preferences into different tenants is performed during the monitoring phase and is an open issue that will be addressed in future work.

5. MODELING

In this section, we describe how the multi-tenant service (Section 5.1) and the user preferences (Section 5.2) are modeled to support the analysis. We use FMs to represent the multi-tenant service including the configuration space (C), the infrastructural resources (I), the workload generated by a service configuration and the users (f_C, f_T), and the obtrusiveness of a service reconfiguration (ρ). We adopt the Semantic Ontology of User Preferences (SOUP) preference model [García et al. 2013] to represent user preferences.

[5]https://wordpress.org/.
[6]http://codex.wordpress.org/Create%20A%20Network.

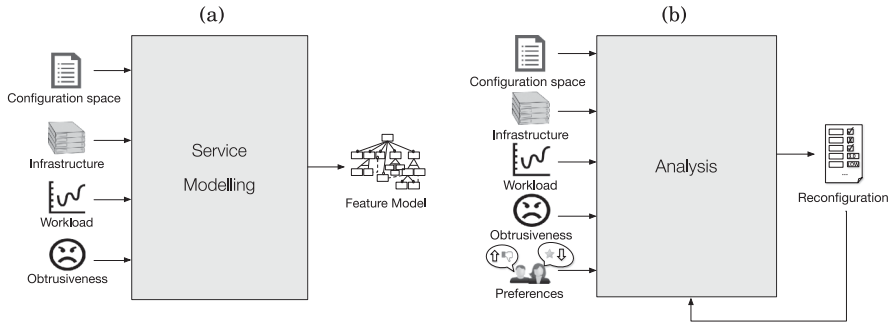

Fig. 3. (a) Service modeling. (b) Analysis inputs and outputs.

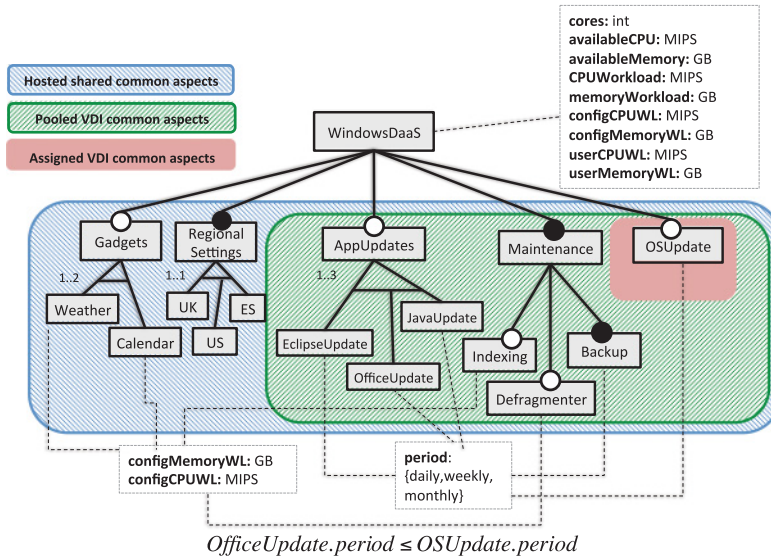

Fig. 4. DaaS configuration space expressed as a feature diagram, - an FM graphical notation.

5.1. Service Modeling

Service modeling usually involves multiple and interrelated configuration options. For example, Amazon EC2 features present more than 20,000 constraints defining 16,991 different configurations [García-Galán et al. 2013]. Our choice to use FMs to model multi-tenant services is motivated by the fact that FMs are expressive enough to represent increasingly complex systems such as cloud services [García-Galán et al. 2013] and content-management frameworks [Sánchez et al. 2014]. This section leverages our DaaS case study to describe how we use FMs to model multi-tenant services.

5.1.1. Configuration Space. Figure 4 shows an FM for the DaaS case study. Each configuration option is modeled either as a feature or as an attribute. Features represent boolean options that can be *selected* or *removed*. Attributes can assume values in an integer, real, or enumerated domain, being suitable to represent non-boolean options. In our case study, the main configuration options are represented by five features: `Gadgets`, `Regional Settings`, `AppUpdates`, `Maintenance`, and `OSUpdates`. These features are in turn decomposed by subfeatures representing possible configuration options. For example, if the `Gadgets` feature is selected, it is necessary to specify whether the `Weather`

forecast gadget, the Calendar gadget, or both are selected. Since the Maintenance feature must be selected mandatorily due to its relationship with the root feature, the Backup feature must also be selected, whereas features Indexing and Defragmenter are optional. For the Backup feature, a daily, weekly, or monthly backup period must be chosen as indicated by the period attribute.

5.1.2. Infrastructure and Workload. In real-world contexts, services have limited infrastructural resources that should be satisfactory to handle the workload determined by the current service configuration and users. We propose to incorporate infrastructure and workload information into the FM by means of attributes and complex constraints. In our example, infrastructure and workload are defined in terms of CPU speed (in MIPS) and RAM size, although other indicators such as incoming and outgoing bandwidth or storage could also be considered. Listing 1 shows an excerpt of the infrastructure and workload definition in the FM using FaMa plain-text notation [Trinidad et al. 2008]. In particular, the syntax for an attribute definition is Feature.attributeName: Domain[range], zero-value;. This indicates the value an attribute assumes when the corresponding feature is removed.

Available infrastructural resources are defined by attributes cores, availableCPU, and availableMemory associated with the WindowsDaas feature. They represent the number of CPU cores and the available CPU and memory and are assigned a fixed value that could only be modified if the infrastructure changes. The attributes representing the overall workload are CPUWorkload and memoryWorkload and are also associated with the WindowsDaas feature. The value of these attributes must always be smaller than the CPU and the RAM available; this is represented in terms of constraints in the FM (first two constraints in Listing 1). The value of these attributes may also vary at runtime depending on the current service configuration and on the number of users.

The overall CPU and RAM workload determined by a service configuration is expressed by the WindowsDaas feature attributes configCPUWL and configMemoryWL, respectively. The value of such attributes is computed as the sum of the CPU and RAM workload determined by each selected leaf feature. This operation is expressed by two constraints in the feature diagram (last two constraints in Listing 1). Each leaf feature is associated with attributes feature.configCPUWL and feature.configMemoryWL representing its CPU and RAM workload, respectively. For example, as shown in Listing 1, the Weather and Indexing features require 30MB and 0.5GB of RAM, respectively, when they are selected. When a feature is removed from a configuration, it does not require resources, and zero values in the attribute definitions are used for this purpose. The workload values associated with each leaf feature can be obtained from real-time data collected while the feature is selected or from estimations when the feature is currently removed.

The CPU and RAM workload determined by the users is represented by the WindowsDaas feature attributes userMemoryWL and userCPUWL, respectively. This workload is usually variable and nonlinear and depends on the user number and profile. Ideally, during the monitoring phase, upper bounds for userMemoryWL and userCPUWL can be predicted. However, since the scope of the solution of this article is on the analysis phase, we use simulated workloads for the example described in Section 6.2.

5.1.3. Obtrusiveness. To model the obtrusiveness of a service configuration, we leverage the conceptual framework proposed by Ju and Leifer [2008]. This framework determines the obtrusiveness level of each interaction of the system with the user by considering the attention dimension; thatis, whether an interaction takes place in the background (the user is unaware of the interaction with the system) or in the foreground (the user is fully conscious of the interaction). Taking inspiration from this work, we consider user awareness about changes as a factor that affects the

```
%Attributes
# Available infrastructure and workload attributes
WindowsDaaS.availableMemory: Real [256]; ## in GBs
WindowsDaaS.availableCPU: Real [400000.0]; ## in MIPS
WindowsDaaS.memoryWorkload: Real [0 to 512];
WindowsDaaS.CPUWorkload: Real [0 to 1000000.0];

# Attributes to compute the current memory workload
WindowsDaaS.userMemoryWL: Real [0 to 256];
WindowsDaaS.configMemoryWL Real [0 to 256];
Weather.configMemoryWL: Real [0.03],0;
Indexing.configMemoryWL: Real [0.5],0;
...
# Attributes to compute the obtrusiveness
Gadgets.obtrusiveness: [3]; ## high obtrusiveness value
Indexing.obtrusiveness: [2]; ## medium obtrusiveness
JavaUpdt.obtrusiveness : [ 1 ] ; ## low obtrusiveness
...
%Constraints
WindowsDaaS.CPUWorkload < WindowsDaaS.availableCPU;
WindowsDaaS.memoryWorkload < WindowsDaaS.availableMemory;
WindowsDaaS.memoryWorkload == WindowsDaaS.configMemoryWL + WindowsDaaS.userMemoryWL;
WindowsDaaS.configMemoryWL == Weather.configMemoryWL + Calendar.configMemoryWL +
Defragmenter.configMemoryWL + Indexing.configMemoryWL + Backup.configMemoryWL + ...;
...
```

Listing 1. Excerpt of DaaS infrastructure, workload, and obtrusiveness modeling for memory using FaMa plaintextformat.

obtrusiveness level of a reconfiguration. In our case, a change is performed when a selected feature is removed, a removed feature is selected, or a configuration option attribute—such as update period—changes value. In this context, the obtrusiveness level produced by changes is the sum of the obtrusiveness level produced by each modified feature. For this reason, we associate an `obtrusiveness` attribute with each feature in the FM. The higher the user awareness about a change in a feature, the higher the `obtrusiveness` of the feature. In particular, those features whose changes affect the graphical user interface, such as `Gadgets` and `Regional Settings`, have high obtrusiveness (3). Features that might cause a slight degradation in system performance, such as `Maintenance`, have medium obtrusiveness (2). For example, `Indexing` is a background task that consumes some CPU and has medium obtrusiveness. Finally, features that are almost transparent to the users, such as `AppUpdates` and `OSUpdates`, have low obtrusiveness (1). Listing 1 shows an example of the obtrusiveness definition for the `Gadgets` and `Indexing` features. In Section 6.3, we explain how these values are used to include obtrusiveness information in the adaptation analysis.

5.2. User Preferences Modeling

In FMs, users can describe their preferences in terms of hard requirements, in which a feature must be either selected or removed and attributes must only assume one specific value in their domains. This approach hinders the negotiation process among different users, making it harder to find a relaxation of conflicting requirements.

Although a service cannot satisfy conflicting hard requirements, it can provide a balance between conflicting preferences. We adapt five preference terms of the SOUP model [García et al. 2013] to express fuzzy user preferences on a given service. SOUP is a highly intuitive and expressive preference model that was initially designed to express preferences for service discovery and ranking. However, it has been adapted to different scenarios, such as resources allocation in business processes [Cabanillas et al. 2013]. We detail the adapted preferences as follows:

—*Favorites* expresses a preference on a selected feature. For example, a user may prefer the `Indexing` feature to be selected.
—*Dislikes* expresses a preference on a removed feature. For example, a user may dislike the `JavaUpdate` feature.
—*Highest* expresses a preference on maximizing the value of a given attribute. For example, a user may prefer the highest value for the `OSUpdate.period` attribute.
—*Lowest* expresses a preference on minimizing the value of a given attribute. For example, a user may want the lowest value for the `JavaUpdate.period` attribute.
—*Around* expresses a preference on a specific attribute value. The closer the attribute value to a target value, the higher the preference satisfaction. For example, a user may want the `OfficeUpdate.period` attribute to be close to the "weekly" value.

In this way, the different users—g rouped by tenants—can employ fuzzy operators to define their satisfaction. Initially, described preference terms were intended to define a partial ranking between a set of services [García et al. 2013]. In our work, we compute the satisfaction of each tenant (i) for each configuration option (j) as a real number $p_{ij} \in [0, 1]$. This choice allows us to measure the preference satisfaction of each tenant i in terms of a fitness function (u_i) described in the next section.

6. ANALYSIS

The goal of our analysis is to identify a service reconfiguration that improves the satisfaction of the users' preferences compared to the current configuration. The analysis takes as input the service model, including the configuration space, the infrastructure and workload, and the features obtrusiveness, the users' preferences model, and the current service configuration, as shown in Figure 3(b). The analysis problem is interpreted as an operation of the AAFM, as described in Section 6.1. A candidate configuration is computed by taking into account the preferences satisfaction and the obtrusiveness determined by its application at runtime. These aspects are considered in the preference-based optimization (Section 6.2) and the obtrusiveness-aware optimization (Section 6.3), respectively.

6.1. Automated Analysis of Feature Models

AAFM is a discipline that deals with "the automated extraction of information from FMs using automated mechanisms" [Benavides et al. 2010]. We leverage existing mappings from FMs to logic paradigms and off-the-shelf solvers to implement our adaptation analysis. In particular, in this article, we use the optimization operation provided by the AAFM framework to perform our adaptation analysis. This operation takes an FM and an objective function as inpu, and returns the configuration fulfilling the criteria established by the function. To optimize the value of the attributes defined in the FMs, relative order preferences have been considered in previous work [Asadi et al. 2014]. However, as far as we are aware, no approach has considered how to optimise fuzzy, high-level user preferences expressed similarly to those described in Section 5.2. Therefore, we have tailored the objective function of the optimization operation of the AAFM framework to support our preference-based optimization.

6.2. Preference-Based Optimization

We interpret our preference-based optimization as a multi-objective constrained optimization problem. From all the available combinations of configuration values, only a subset satisfies all the configuration space, infrastructure, and obtrusiveness constraints for a given time lapse. The set of preferences associated with each tenant is considered as a different objective function; from that subset, it is possible to obtain a Pareto front with solutions that are equally efficient. Table IV shows how preference

Table IV. Preferences Satisfaction

Preference	Element	Satisfaction Measure	Example
Favorites(f)	Feature	$f = selected \Rightarrow p_{ij} = 1$	*Favorites*(Indexing)
Dislikes(f)	Feature	$f = removed \Rightarrow p_{ij} = 1$	*Dislikes*(JavaUpdate)
Highest(att)	Attribute	$p_{ij} = \frac{value - lowerBound}{upperBound - lowerBound}$	*Highest*(OSUpdate.period)
Lowest(att)	Attribute	$p_{ij} = \frac{upperBound - value}{upperBound - lowerBound}$	*Lowest*(JavaUpdate.period)
Around(att,d)	Attribute	$p_{ij} = inverseDistance(value, d)$	*Around*(OfficeUpdate, Weekly)

satisfaction is computed for the analysis. Each preference defines a satisfaction degree p_{ij} comprised between 0 and 1, depending on the value of the element referred to by the preference. The fitness function of each tenant, $u_i = \sum_j p_{ij}$, aggregates its preferences. Since the Pareto front may be composed of a number of equally efficient solutions, we rank them by using a utility function that allows us to choose a single one. To identify an egalitarian solution, we take inspiration from cooperative game theory and the concept of an impartial arbitrator: From two optimal solutions, an impartial arbitrator chooses the most equitable one [Myerson 1991]. Our utility function corresponds to a variation of the Nash product ($\prod_i u_i$), the so-named *Normalized Nash Product*, which compares the different solutions belonging to the Pareto front, as follows:

$$NNP = \prod \frac{u_i \cdot w_i}{U_{iMAX}}$$

where w_i is the number of users of tenant i, and U_{iMAX} is the maximum possible preference satisfaction of each tenant u_i. If different configurations have the same value of the utility function, we select the one minimizing the resources usage. The rest of the section illustrates our analysis approach through the scenario presented in Section 3.

We consider a hosted shared delivery model in which all resources are shared among tenants, as shown in Figure 4. We also consider the preferences of each tenant and their number of users for two subsequent time instants (t and $t + 1$) as shown in Table V. At time t, there are three tenants and the DaaS is running configuration c_1, described in Table VI, which provides satisfaction u_i for each tenant i (Table V).

In the next time instant, a new tenant is added, and the preferences and the number of users associated with each tenant also change. Consequently, the utility value of the current configuration ($u_i(c_1)$) changes accordingly, becoming suboptimal. Therefore, the analysis is triggered, returning a new configuration, c_2 (Table VI), which delivers improved utility values $u_i(c_2)$. The most remarkable improvements are for $tenant_2$ and $tenant_4$, whose preference satisfaction increases from 2.66 to 3.66 and from 0.5 to 1.5, respectively. The improvement of the global satisfaction is also indicated by the value of our utility function (NNP) increasing from 2.2 to 9.1.

As described previously, the available infrastructural resources must be satisfactory to handle the workload generated by the users at each tenant and by the candidate reconfiguration. The workload determined by the users is not linear, and peak MIPS and RAM workloads should be estimated based on the monitored data. For our DaaS scenario, we generate artificial user workload by means of a Gaussian distribution, similarly to Maurer et al. [2013]. For each tenant, we generate a number from a Gaussian distribution, taking $\mu_{CPU} = w_i * AvgCPUWorkload$ and $\mu_{RAM} = w_i * AvgRAMWorkload$—where the average workloads for the CPU and the RAM are extracted from Table I, given the profile and delivery model—and $\sigma_{CPU} = \frac{\mu_{CPU}}{4}$, $\sigma_{RAM} = \frac{\mu_{RAM}}{4}$. The workload generated by the candidate reconfiguration

Table V. Preferences Reconfiguration Scenario* for a Hosted Shared Delivery Model (Changes in Bold)

	t			t+1			
	Preferences	w_i	$u_i(c_1)$	*Preferences*	w_i	$u_i(c_1)$	$u_i(c_2)$
t_1	√ US √ OfficeUpdt Backup.period ≈ Weekly OSUpdt.period ≈ Weekly	45	4	√ US √ OfficeUpdt Backup.period ≈ Weekly OSUpdt.period ≈ Weekly	49	4	4
t_2	¬ UK √ Indexing √ Defragmenter	60	2	¬ UK √ Indexing √ Defragmenter **OfficeUpdt ≈ Monthly**	53	2.66	3.66
t_3	⇑ OfficeUpdt.period ¬ Indexing √ Calendar	31	2.5	⇑ OfficeUpdt.period √ **Defragmenter** √ Calendar	40	2.5	2.5
t_4		0	-	√ **UK** **JavaUpdt.period ≈ Monthly** ⇓ **OfficeUpdt.period**	23	0.5	1.5
				NNP (10^5)		2.2	9.1

*Legend. t_i : $Tenant_i$, √ : Favorites, ¬ : Dislikes, ⇑ : Highest, ⇓ : Lowest, ≈ : Around.

Table VI. Enabled Features and Attribute Values for Configurations c_1 and c_2

	Gadgets	Reg. Set.	App. Updates	Maintenance	OS Update
c_1	Calendar	US	OfficeUpdt.period = *Weekly*	Defragmenter, Backup.period = *Weekly*	OSUpdate.period = *Weekly*
c_2	Calendar	US	OfficeUpdt.period = *Weekly*, JavaUpdt.period = *Monthly*	Defragmenter, Indexing, Backup.period = *Weekly*	OSUpdate.period = *Weekly*

Table VII. Estimation of the Workload Impact on the Infrastructure

		c_1			c_2		
		Workload			*Workload*		
Tenant	*Profile*	w_i	*MIPS*	*RAM* (MB)	w_i	*MIPS*	*RAM* (MB)
T_1	Medium	45	114213	17098	49	71647	28352
T_2	Light	60	74507	10127	53	33953	24303
T_3	Heavy	31	84249	84249	40	154094	37511
T_4	Heavy	-	-	-	23	57136	31316
	Configuration workload		25000	5520		43000	9816
	Total workload		**297969**	**72505**		**359830**	**131298**

is calculated as the sum of the peak workloads (Table II) of each selected configuration option indicated in the last row of Table VI. The total estimated workload is shown in Table VII, as the sum of the tenants workload and the configuration workload. In this case, we assume that the required CPU and memory can be provisioned by the available infrastructural resources depicted in Listing 1.

6.3. Obtrusiveness-Aware Optimization

We characterize the obtrusiveness level of each service reconfiguration as

$$\rho(C_{t+1}, C_t, \Delta t) = \sum_{m \in diff(C_{t+1}, C_t)} \rho_m + max\{\delta_0 - \Delta t, 0\}$$

where $C_{t+1} \in C$ is the candidate configuration; C_t is the current configuration; *diff* is a function that obtains the set of features whose state (selected or removed) or attributes differ between two configurations; ρ_m is the obtrusiveness level assigned for a given feature (defined through the *obtrusiveness* attributes in the FM); Δt is the time elapsed since the last reconfiguration; and δ_0 is the minimum time interval that must pass between two subsequent reconfigurations in order not to disrupt the service usability. δ_0 can be estimated from monitored data.

In this scenario, we add the constraint $\rho < \rho_{MAX}$ to the analysis problem in order to ensure that the obtrusiveness of the reconfigurations from the Pareto front is below a certain threshold. ρ is also set as an additional objective to compute the Pareto front.

For our example scenario, the *diff* function between c_1 and c_2 (Table VI) returns the set {`Indexing`,`JavaUpdt`}. According to the model excerpt of Listing 1, $\rho_m =$ `Indexing.obtrusiveness` + `JavaUpdt.obtrusiveness` $= 2 + 1 = 3$. Considering $\delta_0 = 24$ hours, and $\rho_{MAX} = 4$ hours, at least 20 hours should pass between two subsequent reconfigurations. If $\Delta t = 28$ hours, $\rho = 3 + max\{-4, 0\} = 3 < \rho_{MAX}$, and, therefore, the obtrusiveness of the candidate reconfiguration is below the maximum threshold.

7. EVALUATION

In this section, we illustrate the evaluation of our approach. We describe the implementation of our analysis (Section 7.1) and explain the experiments we conducted to assess its effectiveness and performance (Section 7.2). Finally, we present and discuss our results (Section 7.3).

7.1. Implementation

We have implemented a prototype to perform our preference-based optimization that uses *jMetal*, a Java-based metaheuristics framework to solve multi-objective optimization problems [Durillo and Nebro 2011]. jMetal provides a number of metaheuristic algorithms to compute a Pareto front of the problem. Among all the algorithms that jMetal provides, we chose two genetic algorithms that are widely used for the analysis of FMs [Guo et al. 2011; Sayyad et al. 2013]. Genetic algorithms are search algorithms that work via the process of natural selection. They begin with an initial population of potential solutions, which then evolves through different generations via mutations and crossovers toward a set of more optimal solutions. In particular, we employ *FastPGA* [Eskandari et al. 2007] and *NSGAII* [Deb et al. 2002]. Although both algorithms are elitist, NSGAII establishes different nondomination levels when ranking the—fixed sized—population, whereas FastPGA merges and ranks the previous and current generation into a single—and adaptively sized—population. Due to their complementarity, we decided to compare the behavior of the two algorithms for our analysis. Since the notation we used to describe the configuration space (FaMa plain-text notation [Trinidad et al. 2008]) only supports integer attributes at the moment, we model enumerated domains as an integer range. For the genetic algorithms, the FMs are encoded as an array of boolean variables to represent features selection and integer variables to represent attributes values.

Metaheuristics are partial-search algorithms, and, for this reason, they may consider solutions that violate some constraints of the FM. To avoid this problem, we set the correctness of the solution as an additional objective by taking inspiration from Sayyad et al. [2013]. We measure the violated constraints of a configuration using Choco,[7] a Java CSP solver. The current configuration of the service is taken as input and seeded among the initial population. For the first execution, we seed a random valid

[7]http://www.emn.fr/z-info/choco-solver/.

Table VIII. Amount of Changes Between Two Consecutive Snapshots at $t-1$ and t

Change	t-1	t
#Tenants	$\|T\| = n$	$\|T\| \in \{n-1, n+1\}$
#Prefs	$\|P_i\| = m_i$	$\|P_i\| \in \{m_i - 1, m_i, m_i + 1\}$
#Users	$\|W_i\| = w_i$	$\|W_i\| \in [W_{MIN}, W_{MAX}]$

Table IX. Characteristics of the FMs Used for the Experimental Study

	Features	Atts.	CTC
FM1	18	7	1
FM2	20	14	6
FM3	28	18	9
FM4	29	21	9
FM5	30	21	9

configuration of the service. The intention is twofold: Speed up the generation of valid solutions, and generate some solutions close to the current one. The resulting Pareto front is ranked using our utility function (Section 6). If all the returned points of the Pareto front have a value of the utility function equal to zero ($NNP = 0$) due to each $u_i = 0$, our analysis chooses the solution that maximizes the average satisfaction of the tenants.

7.2. Experiments

Our goal in this experimentation is to check the effectiveness of our analysis. We compare the results obtained using FastPGA and NSGAII with those obtained using a random search algorithm. We measure analysis effectiveness in terms of performed reconfigurations and achieved satisfaction. Performed reconfigurations are measured as the percentage of times the analysis finds a candidate configuration improving the NNP value compared to the current one. Achieved satisfaction is measured as the weighted average of the user preferences satisfaction.

For the experiments, we consider a scenario in which tenants change (i.e., they join and leave the system), and their preferences and number of users vary between different system snapshots, as described in Table V. We define a snapshot as the state of the tenants and their preferences for a specific time instant (t). For every snapshot, we run the analysis to reconfigure the service. We compare the satisfaction achieved by each reconfiguration for the time t to the satisfaction achieved by the previous configuration at the same time.

For our experiments, we define a set of tenants T, each associated with a set of preferences P_i and users W_i. The number of tenants $|T|$ is defined in the integer range $[T_{MIN}, T_{MAX}]$, the number of preferences per tenant i $|P_i|$ is defined in the integer range $[P_{MIN}, P_{MAX}]$, and the number of users $|W_i|$ is defined in the integer range $[W_{MIN}, W_{MAX}]$, considering also that $\sum |W_i| \leq W_{TOTAL}$. For each snapshot (see Table VIII), either one tenant leaves or a new tenant joins the service, but the rest of the tenants may experience changes in their preferences. In particular, if an existing tenant is affected by a change, this can indicate that a new preference is added or an old one is removed. The number of users associated with each tenant (determining its weight) may vary between W_{MAX} and W_{MIN} values. To simulate the changes between consecutive snapshots, we implemented a random generator of tenants and preferences. Given an FM and an integer $k \in [P_{MIN}, P_{MAX}]$, this generator creates T different tenants, each one with a set of different k preferences over features and attributes of the FM. Once a preference has been defined on an element, such element is excluded for future preferences of the same tenant to avoid contradictions. After the initial snapshot is generated, the generator takes as input the set of current tenants and returns a new set of tenants by adding/removing new/existing ones, as shown in Table VIII. It also performs changes in the preferences of the tenants P_i and their number of users w_i.

We consider the configuration space of five services, represented as FMs having increasing complexity. The first FM represents our DaaS scenario in its hosted shared

Table X. Results of the Preference-Based Analysis for FastPGA, NSGAII, and Random Search (RS) Algorithms

	Avg. satisfaction			$NNP(C_t) > NNP(C_{t-1})$			*Avg. execution time (ms)*		
FM	FastPGA	NSGAII	RS	FastPGA	NSGAII	RS	FastPGA	NSGAII	RS
FM1	71.49%	72.15%	62.93%	64.86%	61.98%	30.06%	8254	6293	23704
FM2	56.53%	61.56%	28.01%	41.81%	49.7%	0%	13473	11150	43445
FM3	50.74%	49.3%	30.31%	25.71%	30.05%	0%	17684	15767	63238
FM4	55.01%	64.67%	30.28%	44.25%	47.05%	0%	18138	16035	65049
FM5	45.87%	56.39%	30.83%	24.55%	38.59%	0%	18822	16941	67378

version, and we have employed BeTTy [Segura et al. 2012], a well-known FM generator, to create the remaining FMs. For our evaluation, we assume that the estimated workload can be provisioned by the available infrastructural resources. For instance, the service provider may rely on a third-party infrastructure provider, such as Amazon, which effectively handles elastic provisioning. Table IX shows the characteristics of the generated FMs, where CTC identifies the number of cross-tree constraints (non-hierarchical constraints) of each model. For each FM, we simulated 25 different change scenarios. We randomized the number of snapshots per scenario n in the integer range $[5, 10]$. Initial values and ranges for the remaining parameters are as follows: $T_{min} = 2$, $T_{max} = 5$, $P_{min} = 2$, $P_{max} = 10$, $W_{MIN} = 10$, and $W_{MAX} = 80$. Since each different tenant implies a new objective, we select the same upper limit ($T_{max} = 5$) chosen in related papers on multi-objective optimization for FMs [Sayyad et al. 2013]. We consider $W_{TOTAL} = 200$, since such value is close to the maximum number of users supported by a single real hosted shared DaaS.[8] For the FastPGA and NSGAII algorithms, we rely on the default parameters suggested by jMetal: $Evaluations = 25000$, $PopulationSize = 100$, $CrossoverProbability = 0.9$, and $MutationProbability = 0.05$. For the Random Search algorithm provided by jMetal,[9] we increased the default number of evaluations (25,000) to $Evaluations = 100000$. This algorithm identifies a random configuration with no guarantee that a valid solution satisfying all the constraints is found.

7.3. Experimental Results and Discussion

Table X shows the average satisfaction of the tenant preferences obtained for our experiments, how often a reconfiguration (C_t) improves the NNP value of the previous one (C_{t-1}), and the average execution time for FastPGA, NSGAII, and random search algorithms. We can see that the average satisfaction achieved by the genetic algorithms ranges between 45% and around 70%. Although such satisfaction might not seem to be a good result, we must take into account that the preferences of the different tenants may conflict most of the time, making it infeasible to achieve full satisfaction for such cases. However, genetic algorithms clearly outperform the random search, especially with regard to the achieved average satisfaction.

The percentage of improved reconfigurations—$NNP(C_t) > NNP(C_{t-1})$—ranges between 25% and 65% for FastPGA and between 30% and 62% for NSGAII. Although at a first glance this may seem a poor result, it is necessary to consider that this number highly depends on changes between consecutive system snapshots. The greater the changes, the greater the chances to decrease the satisfaction of the previous configuration and the greater the chances to find an improved reconfiguration. In addition, since we look for egalitarian solutions, our algorithms discard solutions that may have

[8]Using 2xE5-2470 2.3GHz processors, IBM was able to support 206 users: http://blogs.citrix.com/2013/10/29/extreme-density-5768-hosted-shared-desktops-in-a-single-blade-chassis/.

[9]http://jmetal.sourceforge.net/javadoc/jmetal/metaheuristics/randomSearch/RandomSearch.html.

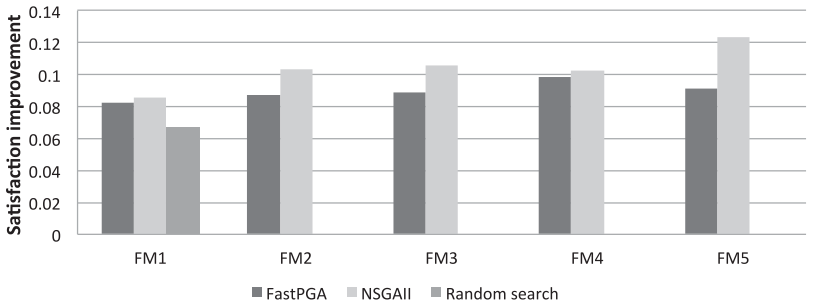

Fig. 5. Average satisfaction improvement with respect to the previous configuration.

a better average satisfaction but ignore the preferences of particular tenants (i.e., one of the tenants has 0 satisfaction, which leads to NNP = 0). Both genetic algorithms perform better than the random search, which cannot return any improved reconfiguration in most of the cases and whose execution time is about four time higher. This is because in a constrained optimization problem we need to consider the constraints in order to return valid solutions. Although we could add the correctness to the solution as an additional objective for the genetic algorithms, this was not possible for the random search. The execution time for the genetic algorithms is between 6 and 19 seconds, suggesting that our analysis can be performed at runtime.

Figure 5 indicates the average satisfaction improvement for the successful reconfigurations, which range between 8% and 12% in absolute terms (i.e., the worst result returns 0% satisfaction and the perfect result in a conflict-free scenario returns 100% satisfaction). In general terms, NSGAII algorithm performs better than FastPGA, especially with larger models: Except for FM1, NSGAII outperforms the rate of improvements obtained using FastPGA. For the first FM, the random search algorithm performs worse than the genetic algorithms but better than its own behavior for the rest of the FMs. This is due to the fact that the configuration space of the first model is smaller than the other FMs and allows the random search to find some acceptable solutions.

8. OPEN ISSUES

In this article, we do not address all the challenges necessary to support user-centric adaptation of multi-tenant services. In this section, we describe the open issues by grouping them depending on which activities of the MAPE loop they belong.

(a) **Monitoring**: In this work, we assume that the monitoring phase is able to obtain all the data required for the analysis. However, for a comprehensive approach, we need to propose specific ways to extract user preferences, monitor the workload determined by the service configurations and the users, and measure user satisfaction and reconfiguration obtrusiveness (e.g., by means of empirical studies). Modifications in the configurations determined by system changes or updates should also be detected and monitored.
(b) **Analysis**: One of the main limitations of the analysis is the simplicity of our workload estimation. In future work, it will necessary to use monitored data to predict resources usage determined by a specific configuration and users' profiles; additional aspects, such as thrashing, should also be incorporated. Moreover, we recognise that although our assumption of uniformity within tenant groups is simplistic, it was very useful to develop a first version of our analysis approach.

However, in future work, we will use a more precise and updated model of user behavior and preferences within each tenant.

(c) **Planning**: The reconfiguration actions of the service must be planned systematically in order to avoid inconsistent service states and user interruption.

(d) **Execution**: A reconfiguration engine on the specific service—a DaaS in our case—remains to be implemented in order to execute planned configuration changes.

Furthermore, other aspects may threaten the validity of our approach:

—**Malicious Users.** A malicious user may intentionally express specific preferences to achieve a desired service configuration. There are two possible ways in which this problem can be avoided. An option could be to exclude critical features from the adaptation process. This is why in our case study we did not allow users to express preferences on security features such as firewall level or antivirus update frequency. Another option is to consider a preference in the analysis only if at least a certain number of users has expressed it.

—**Preferences Aggregation Function.** The NNP function we adopt to aggregate user preferences tries to balance user satisfaction, thus avoiding configurations that deliver a 0-valued satisfaction to any tenant. However, this may result in an unfair adaptation when, for example, a tenant with a single user that expresses a particular preference gets it satisfied while decreasing the satisfaction of the rest of the tenants with multiple users. To address this issue, we can modify the NNP function by assigning a weight to a specific preference depending on the number of users sharing it.

9. RELATED WORK

The idea of performing adaptations to improve user satisfaction is not new. Other approaches [Malek et al. 2012; Cardellini et al. 2012] have considered conflicting users' preferences and limited infrastructural resources in the construction of adaptive software systems. For example, Malek et al. [2012] propose the redeployment of software components on hardware nodes in order to optimize conflicting QoS dimensions for changing user preferences. Cardellini et al. [2012] present a reference framework for self-adaptation of service-oriented systems in which user satisfaction is considered as an adaptation driver. Both approaches solve an optimization problem to reconfigure a system architecture and a service-oriented application, respectively. The optimization problem is solved by means of different techniques, such as integer programming or genetic algorithms. Ali et al. [2012] propose social adaptation, which aims to dynamically adapt existing software systems depending on user collective judgment expressed on the way the system should behave. This approach treats user feedback as a primary driver for planning and guiding adaptation. Feedback is related to the selection of a specific system feature when more than one of them can be enabled. This work also provides an analysis activity to select a feature configuration that fulfils user preferences. Differently from these approaches, we propose user-centric adaptation for the reconfiguration of multi-tenant services in cloud scenarios where users can come and go, and different service configurations and resources are shared depending on the cloud service model. Dalpiaz et al. [2012] propose leveraging the preferences of non-functional requirements expressed by a single user as a key driver for adaptation. The collected preferences are used to adapt the selection of routine tasks to be performed in a pervasive infrastructure by a user. In contast, we focus on the maximization of the satisfaction of the global user preferences expressed on shared service configurations. Song et al. [2013] present an approach to develop self-adaptive systems that take into account end-users who express their preferences on the adaptation actions selected by the system in order to better tune the adaptation results. Our work, instead, has a

different focus since we consider changes of user preferences as one of the main triggers for adaptation and aim to improve the global satisfaction of user preferences.

Lamparter et al. [2007] propose an ontology for representing and matching configurable web services. In particular, service configurations and associated preferences are represented using function policies that allow the characterizing of service attributes and the semantics and price of their value. The authors also propose an optimal algorithm for service selection based on linear programming. Differently from this work, we do not focus on the preferences expressed by a single user. Furthermore, we do not explicitly represent price, but we assume that service providers can satisfy user preferences up to a maximum amount of resources that can be allocated. A different approach is adopted by Gallacher et al. [2013] who propose an algorithm to learn contextual user preferences without explicitly asking for feedback in order to drive personal adaptations. The authors try to overcome problems related to the accuracy of the preferences even when input sources come and go and users change their behavior. This work focuses on improving the preference satisfaction of a single user, whereas we assume that user preferences are given and address the problem of maximizing their global satisfaction.

Cloud services analysis and adaptation has been a prolific research area during the past years. Caton and Rana [2012] propose an approach for cloud infrastructure provisioning through volunteered resources. It relies on autonomic fault management techniques to adapt resource usage. In this direction, Maurer et al. [2013] also propose an adaptive resource configuration for cloud infrastructure management. In this case, they structure adaptation actions into levels and rely on Case-Based Reasoning and a rule-based approach in order to counteract SLA violations. Wei et al. [2010] present a similar idea, with the difference that they intend to reach an equilibrium among resources allocation performed on behalf of different users. To achieve this aim, Wei et al. use a game theoretic approach based on Nash equilibria. Pitt et al. [2012] also propose a resource allocation method that is focused on the notion of fairness for the agents who share a common pool of resources. The authors take inspiration from the principles of enduring institutions [E. Ostrom 1990] to identify a resource allocation method based on canons of distributive justice. In particular, they propose a variant of the Linear Public Good game as an abstract representation of the resource allocation scenarios found in ad-hoc networks, sensor networks, and smart grids. This approach demonstrates a way to produce a better balance of utility and fairness. Differently from us, the approaches [Caton and Rana 2012; Maurer et al. 2013; Wei et al. 2010; Pitt et al. 2012] presented previously focus on resource allocation instead of feature selection and maximization of global user preferences. Finally, Vankeirsbilck et al. [2014] propose a model for identifying an optimal resource allocation in order to satisfy virtual desktop requests based on the tradeoff between costs and revenues for the service provider. The authors also consider the possibility of overbooking (i.e., the probability that fewer resources are allocated for the virtual desktops than are needed). This approach is agnostic of user preferences and takes into account only resource allocation as a measure of SLA violations.

Other works are more focused on cloud adaptation at the application level. Inzinger et al. [2013] propose a model-based adaptation that allows cloud application developers to specify behavior goals and adaptation rules. These models are "management hooks" for the cloud providers who can implement an adaptation strategy by considering the preferences of multiple customers and low-level infrastructural constraints. Marquezan et al. [2014] provide a conceptual model that characterizes all entities of the cloud environment that are relevant for adaptation decisions, the concrete adaptation mechanisms and actions that these entities may perform, and the mutual dependencies between these entities and actions. This allows cloud developers to take informed

decisions on which kind of adaptation mechanisms to exploit for their application. Differently from this work, in our approach, we model user preferences expressed over the service configurations and identify infrastructural resources required to support specific service delivery models. These models allows us to configure our user preference analysis automatically.

Nallur and Bahsoon [2013] propose an adaptive mechanism for applications composed of different cloud services. Adaptation dynamically selects the best value-for-money services and is triggered by violations of QoS by any of the adopted concrete services and by changes of an application target QoS. The authors propose an approach based on Market-Based Control (MBC) to self-adaptation: Bids are the mechanism by which the search space of QoS-cost combinations is explored. This approach is mainly focused on service selection rather than features selection and aims to maximize the satisfaction of a single user. Our aim instead is to maximize global user preferences and to minimize the obtrusiveness of adaptation. A security-oriented perspective is instead assumed by Bernabe et al. [2014] who propose an advanced authorization model that provides conditional role-based access control. This adapts the privileges assigned to roles depending on the groups of tenants sharing the same resources.

Research on SPLs is highly related to our article. The idea of using variability techniques to model the adaptation space is not new. For example, Bencomo et al. [2008] propose the use of variability modeling to define the runtime adaptation space. Concerning multi-tenancy and SPLs, Schroeter et al. [2012a, 2012b] use variability and SPLs techniques to assist the configuration of multi-tenant applications. The authors identify configuration requirements and propose a configuration process using FMs [Schroeter et al. 2012b], and they also define requirements and middleware for a variable multi-tenant architecture [Schroeter et al. 2012a]. However, this work has not considered how to reconfigure multi-tenant applications at runtime, when user preferences, available service configurations, and infrastructural constraints change.

Mietzner et al. [2009] propose the use of variability modeling techniques to manage the variability of SaaS applications and their requirements. Specifically, they use variability models to configure and deploy SaaS applications for different tenants. However, they focus on modeling the variability and deploying different variants of an SaaS application instance. Variability of different cloud providers has also been analyzed and modeled by García-Galán et al. [2013] to assist the migration of an in-house infrastructure to the cloud. However, this approach works with hard requirements and ignores changes of user preferences. Similarly to us, Stein et al. [2014] consider the problem of configuring multi-tenant services in order to better satisfy tenant preferences on average. Based on such preferences, different product configurations using different strategies from the social theory are suggested. However, preferences are only expressed as hard and soft constraints, and the analysis does not take into account infrastructural constraints that might prevent the satisfaction of users' preferences. Furthermore, the approach is not adapted to support runtime reconfiguration, and, for this reason, feasibility of the proposed analysis at runtime has not been investigated.

Several research efforts have been made to investigate multi-objective optimization in applications characterized by variability. Guo et al. [2011] use a genetic algorithm to find optimal FM configurations for a single objective and user. Sayyad et al. [2013] perform multi-objective optimization on several large FMs using metaheuristics. However, their objective functions are fixed (i.e., minimize errors and cost or maximize number of features), whereas our fitness function depends on specific user preferences. Finally, other work has explored techniques for solving conflicts in a configuration process. White et al. [2010] propose a technique in this direction that only considers a single user and a minimal changes criterion. Although García-Galán et al. [2013] consider

multiple users, after detecting the conflicts, these users have to define explicitly the impact of every solution on their preferences satisfaction.

10. CONCLUSION AND FUTURE WORK

In this article, we present an approach to support user-centric adaptation of multi-tenant services. We motivate our proposal using a multi-tenant DaaS case study and explain how to engineer the activities of the MAPE loop necessary to support user-centric adaptation. Our focus is on the analysis activity of the MAPE loop that identifies a service reconfiguration that maximizes tenants' preferences satisfaction. The analysis also guarantees that the computed service reconfiguration can be provisioned by using the infrastructural resources available at the provider side. This analysis takes as input the model of the service and user preferences. We use FMs to model the multi-tenant service, which, more precisely, describes the service configuration space, the infrastructural constraints, the workload, and the obtrusiveness of the configurations. We adopt the SOUP preference model to represent user preferences.

The analysis is interpreted as a multi-objective constrained optimization problem, where the different objectives are defined by the preferences of the tenants. This optimization problem is solved using genetic algorithms (FastPGA and NSGAII) that identify the Pareto front of potential candidate reconfigurations. Obtained experimental results demonstrate that our analysis approach (i) identifies reconfigurations that improve user satisfaction and (ii) can be performed at runtime. FastPGA provides more effective results for smaller FMs, whereas NSGAII is more effective when bigger FMs have to be analyzed.

As future work, we are planning to evaluate the applicability of our approach with practitioners by using real multi-tenant services. This will require collecting experimental data related to the impact that specific service features have on the consumption of the infrastructural resources. Regarding the whole user-centric adaptation problem, we will integrate our analysis with other activities of the MAPE loop. In particular, for the monitoring activity, we will leverage existing work [Gallacher et al. 2013] to measure user preferences in a nonintrusive and precise way. For the planning and execution activities, we will adopt real multi-tenant services to identify possible strategies to enact a service reconfiguration on the system at runtime depending on the current configuration and the number of users. Finally, we are planning to conduct empirical studies to estimate more precisely how adaptation obtrusiveness is perceived by real users.

ACKNOWLEDGMENTS

We want to thank the anonymous reviewers for their comments, which were very beneficial in improving the quality of this work significantly. We are also grateful to J. M. García for the explanations on user preferences modeling, and to J. A. Parejo and A. Sayyad for their advice on experimentation.

REFERENCES

Raian Ali, Carlos Solis, Inah Omoronyia, Mazeiar Salehie, and Bashar Nuseibeh. 2012. Social adaptation: When software gives users a voice. In *Proceedings of the 7th International Conference on the Evaluation of Novel Approaches to Software Engineering*.

Mohsen Asadi, Samaneh Soltani, Dragan Gasevic, Marek Hatala, and Ebrahim Bagheri. 2014. Toward automated feature model configuration with optimizing non-functional requirements. *Information and Software Technology* 56, 9 (2014), 1144–1165.

Luciano Baresi, Sam Guinea, and Liliana Pasquale. 2012. Service-oriented dynamic software product lines. *IEEE Computer* 45, 10 (2012), 42–48.

David Benavides, Sergio Segura, and Antonio Ruiz-Cortes. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615–636.

Nelly Bencomo, Peter Sawyer, Gordon S. Blair, and Paul Grace. 2008. Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems. In *Proceedings of the 12th International Conference on Software Product Lines (Workshops)*. 23–32.

Jorge Bernal Bernabe, Juan M. Marin Perez, Jose M. Alcaraz Calero, Felix J. Garcia Clemente, Gregorio Martinez Perez, and Antonio F. Gomez Skarmeta. 2014. Semantic-aware multi-tenancy authorization system for cloud architectures. *Future Generation Computer Systems* 32 (2014), 154–167.

C.-P. Bezemer, Andy Zaidman, Bart Platzbeecker, Toine Hurkmans, and A. T. Hart. 2010. Enabling multi-tenancy: An industrial experience report. In *Proceedings of the 26th International Conference on Software Maintenance*. 1–8.

Cristina Cabanillas, José M. García, Manuel Resinas, David Ruiz, Jan Mendling, and A. Ruiz-Cortés. 2013. Priority-based human resource allocation in business processes. In *Proceedings of the 11th International Conference on Service-Oriented Computing*. 374–388.

Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Stefano Iannucci, Francesco Lo Presti, and Raffaela Mirandola. 2012. Moses: A framework for QoS driven runtime adaptation of service-oriented systems. *IEEE Transactions on Software Engineering* 38, 5 (2012), 1138–1159.

Simon Caton and Omer Rana. 2012. Towards autonomic management for cloud services based upon volunteered resources. *Concurrency and Computation: Practice and Experience* 24, 9 (2012), 992–1014.

Citrix. 2013. Citrix Virtual Desktop Handbook 7.x. Retrieved from http://support.citrix.com/article/CTX139331.

Fabiano Dalpiaz, Estefan'ia Serral, Pedro Valderas, Paolo Giorgini, and Vicente Pelechano. 2012. A NFR-based framework for user-centered adaptation. In *Proceedings of the 31st International Conference on Conceptual Modeling*, 439–448.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

Juan J. Durillo and Antonio J. Nebro. 2011. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software* 42, 10 (2011), 760–771.

E. Ostrom. 1990. *Governing the Commons*. CUP.

Hamidreza Eskandari, Christopher D. Geiger, and Gary B. Lamont. 2007. FastPGA: A dynamic population sizing approach for solving expensive multiobjective optimization problems. In *Proceedings of the 4th International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 141–155.

Krzysztof Z. Gajos, Mary Czerwinski, Desney S. Tan, and Daniel S. Weld. 2006. Exploring the design space for adaptive graphical user interfaces. In *Proceedings of the International Working Conference on Advanced Visual Interfaces.*, 201–208.

Sarah Gallacher, Eliza Papadopoulou, Nick K. Taylor, and M. Howard Williams. 2013. Learning user preferences for adaptive pervasive environments: An incremental and temporal approach. *ACM Transactions on Autonomous Adaptive Systems* 8, 1 (2013).

José M. García, Martin Junghans, David Ruiz, Sudhir Agarwal, and Antonio Ruiz-Cortés. 2013. Integrating semantic web services ranking mechanisms using a common preference model. *Knowledge-Based Systems* 49 (2013), 22–36.

Jesús García-Galán, Liliana Pasquale, Pablo Trinidad, and Antonio Ruiz Cortés. 2014. User-centric adaptation of multi-tenant services: Preference-based analysis for service reconfiguration. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 65–74.

J. García-Galán, O. F. Rana, P. Trinidad, and A. Ruiz-Cortés. 2013. Migrating to the cloud: A software product line based analysis. In *Proceedings of the 3rd International Conference on Cloud Computing and Services Science*. 416–426.

Jesús García-Galán, Pablo Trinidad, and Antonio Ruiz-Cortés. 2013. Multi-user variability configuration: A game theoretic approach. In *Proceedings of the 28th International Conference on Automated Software Engineering*. 574–579.

Jianmei Guo, Jules White, Guangxin Wang, Jian Li, and Yinglin Wang. 2011. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software* 84, 12 (2011), 2208–2221.

Christian Inzinger, Benjamin Satzger, Philipp Leitner, Waldemar Hummer, and Schahram Dustdar. 2013. Model-based adaptation of cloud computing applications. In *Procedings of the 1st International Conference on Model-Driven Engineering and Software Development*. 351–355.

Wendy Ju and Larry Leifer. 2008. The design of implicit interactions: Making interactive systems less obnoxious. *Design Issues* 24, 3 (2008), 72–84.

Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-021. Software Engineering Institute, Carnegie Mellon University.

J. O. Kephart and D. M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50.

I. Kumara, J. Han, A. Colman, Tuan Nguyen, and M. Kapuruge. 2013. Sharing with a difference: Realizing service-based saas applications with runtime sharing and variation in dynamic software product lines. In *Proceedings of the 10th International Conference on Services Computing*. 567–574.

Steffen Lamparter, Anupriya Ankolekar, Rudi Studer, and Stephan Grimm. 2007. Preference-based selection of highly configurable web services. In *Proceedings of the 16th International Conference on World Wide Web*. 1013–1022.

Sam Malek, Nenad Medvidovic, and Marija Mikic-Rakic. 2012. An extensible framework for improving a distributed software system's deployment architecture. *IEEE Transactions on Software Engineering* 38, 1 (2012), 73–100.

R. Timothy Marler and Jasbir S. Arora. 2004. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization* 26, 6 (2004), 369–395.

Clarissa Cassales Marquezan, Florian Wessling, Andreas Metzger, Klaus Pohl, Chris Woods, and Karl Wallbom. 2014. Towards exploiting the full adaptation potential of cloud applications. In *Proceedings of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems*, 48–57.

Michael Maurer, Ivona Brandic, and Rizos Sakellariou. 2013. Adaptive resource configuration for cloud infrastructure management. *Future Generation Computer Systems* 29, 2 (2013), 472–487.

Ralph Mietzner, Andreas Metzger, Frank Leymann, and Klaus Pohl. 2009. Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In *Proceedings of the International Workshop on Principles of Engineering Service-Oriented Systems*, 18–25.

Roger B. Myerson. 1991. *Game Theory: Analysis of Conflict*. Harvard University Press.

V. Nallur and R. Bahsoon. 2013. A decentralized self-adaptation mechanism for service-based applications in the cloud. *IEEE Transactions on Software Engineering* 39, 5 (2013), 591–612.

Yefim V. Natis. 2012. *Gartner Reference Model for Elasticity and Multitenancy*. Technical Report. Gartner, Inc.

J. Pitt, J. Schaumeier, D. Busquets, and S. Macbeth. 2012. Self-organising common-pool resource allocation and canons of distributive justice. In *Proceeddings of the 6th International Conference on Self-Adaptive and Self-Organizing Systems*. 119–128.

Ana B. Sánchez, Sergio Segura, and Antonio Ruiz-Cortés. 2014. The drupal framework: A case study to evaluate variability testing techniques. In *Proceedings of the 8th International Workshop on Variability Modelling of Software-intensive Systems*. 11.

Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Scalable product line configuration: A straw to break the camel's back. In *Proceedings of the 28th International Conference on Automated Software Engineering*. 465–474.

Julia Schroeter, Sebastian Cech, Sebastian Götz, Claas Wilke, and Uwe Assmann. 2012a. Towards modeling a variable architecture for multi-tenant SaaS-applications. In *Proceedings of the 6th Workshop on Variability Modeling of Software-Intensive Systems*.

Julia Schroeter, Peter Mucha, Marcel Muth, Kay Jugel, and Malte Lochau. 2012b. Dynamic configuration management of cloud-based applications. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*, 171–178.

S. Segura, José Á. Galindo, David Benavides, José A. Parejo, and A. Ruiz-Cortés. 2012. BeTTy: Benchmarking and testing on the automated analysis of feature models. In *Proceedings of the 6th Workshop on Variability Modeling of Software-Intensive Systems*. Leipzig, Germany.

Hui Song, Stephen Barrett, Aidan Clarke, and Siobhán Clarke. 2013. Self-adaptation with end-user preferences: Using run-time models and constraint solving. In *Proceedings of the 16th International Conference on Model-Driven Engineering Languages and Systems*. 555–571.

Cheri Speier, Iris Vessey, and Joseph S. Valacich. 2003. The effects of interruptions, task complexity, and information presentation on computer-supported decision-making performance. *Decision Sciences* 34, 4 (2003), 771–797.

Jacob Stein, Ingrid Nunes, and Elder Cirilo. 2014. Preference-based feature model configuration with multiple stakeholders. In *Proceedings of the 18th International Software Product Lines Conference*. 132–141.

Thomas Thum, Christian Kastner, Sebastian Erdweg, and Norbert Siegmund. 2011. Abstract features in feature modeling. In *Proceedings of the 15th International Software Product Line Conference*. 10.

P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez. 2008. FAMA framework. In *12th Software Product Lines Conference*. 359–359.

Bert Vankeirsbilck, Lien Deboosere, Pieter Simoens, Piet Demeester, Filip De Turck, and Bart Dhoedt. 2014. User subscription-based resource management for desktop-as-a-service platforms. *Journal of Supercomputing* 69, 1 (2014), 412–428.

Guiyi Wei, Athanasios V. Vasilakos, Yao Zheng, and Naixue Xiong. 2010. A game-theoretic method of fair resource allocation for cloud computing services. *Journal of Supercomputing* 54, 2 (2010), 252–269.

J. White, D. Benavides, D. C. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes. 2010. Automated diagnosis of feature model configurations. *Journal of Systems and Software* 83, 7 (2010), 1094–1107.

Received October 2010; revised March 2015; accepted June 2015