User-Centric Adaptation of Multi-tenant Services: Preference-Based Analysis for Service Reconfiguration

Jesús García-Galán1, Liliana Pasquale2, Pablo Trinidad3, Antonio Ruiz-Cortés1
1University of Seville, Seville, Spain
2Lero - the Irish Software Engineering Research Centre, University of Limerick, Ireland

ABSTRACT
Multi-tenancy is a key pillar of cloud services. It allows different tenants to share computing resources transparently and, at the same time, guarantees substantial cost savings for the providers. However, from a user perspective, one of the major drawbacks of multi-tenancy is lack of configurability. Depending on the isolation degree, the same service instance and even the same service configuration may be shared among multiple tenants (i.e., shared multi-tenant service). Moreover tenants usually have different - and in most of the cases - conflicting configuration preferences. To overcome this limitation, this paper introduces a novel approach to support user-centric adaptation in shared multi-tenant services. The adaptation objective aims to maximise tenants' satisfaction, even when tenants and their preferences change during the service life-time. This paper describes how to engineer the activities of the MAPE loop to support user-centric adaptation, and focuses on the analysis of tenants' preferences. In particular, we use a game theoretic analysis to identify a service configuration that maximises tenants' preferences satisfaction. We illustrate and motivate our approach by utilising a multi-tenant desktop scenario. Obtained experimental results demonstrate the feasibility of the proposed analysis.

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
Human Factors, Measurement

Keywords
Adaptive systems, multi-tenancy, cloud, game theory

1. INTRODUCTION
Multi-tenancy [26] allows cloud providers to deliver the same service to different tenants, which share physical and/or virtual resources transparently. Depending on the adopted cloud model, tenants can share resources at 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 larger the resources and cost savings, but the smaller the configurability. The shared multi-tenant model offers the lowest degree of isolation, since the same service instance is shared by all the tenants. Such model guarantees great resources’ savings, but neglects configuration capabilities, since even minimal configuration changes may have an impact on all the users. From the user perspective, this lack of configurability [22] is a major drawback of multi-tenancy, especially when users’ preferences are not known in advance. Several approaches [2, 19, 22, 29, 30] have been proposed to support dynamic configuration management in multi-tenant services. Nonetheless, these contributions consider an isolated multi-tenant model, and they focus on deploying different variants of existing service instances at runtime.

Recently, the notion of social adaptation [1] has been proposed to promote users as first class entities in the adaptation process. In particular, social adaptation considers changes in the users’ collective judgement as a new adaptation driver. However, additional challenges have to be addressed to fully support adaptation for shared multi-tenant services. First, tenants often have different and conflicting 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 may affect the global tenants’ preferences. Finally, the underlying software and hardware infrastructure has a limited capacity, which may jeopardise the satisfaction of the tenants’ preferences. Some of these challenges have been addressed by the software engineering community. In particular, requirements prioritisation [6, 18] and cloud infrastructure management techniques [21] provide solutions to tradeoff among conflicting stakeholders’ preferences. However, to our knowledge, none of the existing approaches provides a framework to support user-centric adaptation in order to maximise the satisfaction of tenants’ preferences at runtime.

Our proposal has the objective to support dynamic user-centric adaptation in shared multi-tenant services, where tenants’ preferences may change at runtime and may conflict among each other. We motivate and explain the ideas presented in this paper by using a running example of a
multi-tenant Desktop as a Service (DaaS). Since we assume a shared multi-tenant model, our approach is aimed to balance the tradeoff between tenants’ satisfaction and adaptation transparency. On the one hand, the adaptation should maximise the satisfaction of the tenants’ preferences, which are expressed on the configurable elements of the system. On the other hand, the adaptation should not be too intrusive i.e., the adaptation actions should have a minimal impact on the usability of the system.

The contribution of this paper is twofold. First, we define the user-centric adaptation problem in terms of challenges and activities of the MAPE (Monitoring, Analysis, Planning, Execution) loop necessary to support it. Second, we propose a preference-based analysis for identifying service configurations that maximise the tenants’ satisfaction at runtime. We express the configuration space in terms of an Extended Feature Model (EFM) and the tenants’ preferences by means of a preferences model [13]. The analysis is interpreted as a coalitional problem of game theory [25]. We have implemented a prototype to perform the proposed preferences analysis by leveraging metaheuristics for multi-objective optimisation [20], which have been proved suitable for EFM optimisation in existing work [16, 28, 36]. Obtained experimental results demonstrate that our preference-based analysis effectively maximises tenants’ preferences and can be performed in a short time.

The rest of the paper is organised as follows. Section 2 describes the motivating scenario of a shared multi-tenant service. Section 3 illustrates the general user-centric adaptation problem, and the specific preferences-based analysis. Section 4 describes the proposed solution and Section 5 discusses experimental results. Section 6 compares our approach with relevant related work and Section 7 concludes.

2. MOTIVATING SCENARIO

Multi-tenancy is defined as “multiple users or processes (tenants) sharing common physical or virtual computing resources while remaining logically independent” [26]. Multi-tenancy is considered an essential characteristic of cloud computing, as it allows service providers to support elastic resource provisioning and pay-as-you-go billing models. However, such benefits often collide with security and configurability concerns. The more the shared resources, the higher the security concerns and the lower the configurability. On the one hand, limited configurability may have an impact on the users’ satisfaction, but, on the other hand, granting users unlimited configurability options - on a service which is shared by different tenants - may even have a worse impact on users’ satisfaction. For this reason, providers support different degrees of multi-tenancy, depending on factors like the users’ preferences, available infrastructure, and the characteristics of the delivered service.

In this paper we have chosen the specific case of a Desktop as a Service (DaaS) as a motivating scenario. DaaS provides a virtual desktop and a set of applications-as-a-service to a single or multiple tenants. Providers like Citrix1, VMWare2, and Amazon3 are increasingly offering a wide range of DaaS solutions. Common DaaS delivery models include Virtual Desktop Infrastructure (VDI) and hosted shared model. The VDI model delivers a different desktop instance to each tenant, while the hosted shared model provides a different session of the same desktop instance to each tenant. These two models present benefits and drawbacks. VDI provides a more configurable DaaS with higher costs, while, despite the hosted shared model is cheaper, it has the limitation of not being flexible enough, since some tenants’ preferences are intentionally neglected to avoid potentially negative side-effects on the satisfaction of the other tenants.

Table 1: Tenants’ preferences (conflicting preferences are indicated by using the same number).

<table>
<thead>
<tr>
<th>Preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tenant 1</td>
</tr>
<tr>
<td>- Aero preferred over Classic (1)</td>
</tr>
<tr>
<td>- Frequent Latex and Office updates</td>
</tr>
<tr>
<td>- Frequent antivirus checks (2)</td>
</tr>
<tr>
<td>- Highest firewall level (3)</td>
</tr>
<tr>
<td>Tenant 2</td>
</tr>
<tr>
<td>- Classic preferred over Aero (1)</td>
</tr>
<tr>
<td>- Indexing and defragmentation</td>
</tr>
<tr>
<td>- Medium firewall level (3)</td>
</tr>
<tr>
<td>- No OS updates</td>
</tr>
<tr>
<td>Tenant 3</td>
</tr>
<tr>
<td>- Medium frequency for office updates</td>
</tr>
<tr>
<td>- Frequent backups (4)</td>
</tr>
<tr>
<td>- Unfrequent antivirus checks (2)</td>
</tr>
<tr>
<td>- Medium firewall level (3)</td>
</tr>
<tr>
<td>Tenant 4</td>
</tr>
<tr>
<td>- Aero preferred over Classic (1)</td>
</tr>
<tr>
<td>- Frequent office updates</td>
</tr>
<tr>
<td>- Slightly frequent backups (4)</td>
</tr>
<tr>
<td>- Very frequent antivirus checks (2)</td>
</tr>
<tr>
<td>- Highest firewall level (3)</td>
</tr>
</tbody>
</table>

As a concrete example scenario, we consider a corporate organisation that uses a hosted shared DaaS. For this scenario we have four different tenants representing the preferences of their corresponding users. Each tenant exhibits a different set of preferences (Table 1). For example, while tenants 2 and 3 prefer a medium firewall level, tenants 1 and 4 prefer the highest firewall level (conflict 1). Similarly, tenant 1 prefers Aero over Classic look & feel, while tenant 2 has a completely opposite preference. The first challenge of this scenario is to maximise the satisfaction of the tenants’ preferences. Note that the complete satisfaction of all tenants’ preferences is infeasible in most of the cases, and therefore it is necessary to tradeoff between them.

Similarly to other cloud service models, DaaS satisfies the requests of its tenants elastically. This means that new users and tenants can join the system or existing ones can leave dynamically. For example, users who belong to tenant 3 work at fixed times, while the rest of the tenants accesses the DaaS at different times (including weekends), especially when project deadlines are close. Similarly, the current DaaS configuration may become sub-optimal because tenants’ preferences vary during the system life-time. For example, some users of tenant 2 may prefer a lower firewall and antivirus level to run some tests. In all these cases, the current users of the DaaS and their preferences must have a direct impact on the selection of a specific service configuration. Therefore, the second challenge of this scenario is to be able to adapt the DaaS and - more in general - multi-tenant services dynamically, when user-related changes take place.

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

In this section, we provide a big picture of the user-centric adaptation problem (Subsection 3.1). Then, we describe the preference-based analysis, which is the specific challenge addressed in this paper (Subsection 3.2).

![Diagram of the MAPE loop for user-centric adaptation](image)

**Figure 1: User-Centric Adaptation Loop.**

### 3.1 Engineering User-centric Adaptation

We define user-centric adaptation as the adaptation process aimed to continue to maximise users’ satisfaction at runtime, even when the operational environment changes. These changes can affect users’ preferences, available system configurations, and computational resources. As previously proposed by Ali et al. [1], users are considered as first class entities during runtime adaptation. In this paper adaptation is performed when any event that may have an impact on the users’ satisfaction is detected, such as new/existing users joining/leaving the system or modifications of users’ preferences on specific system configurations. For the specific scenario proposed in this paper, user-centric adaptation aims to balance the tradeoff between satisfaction of tenants’ preferences on the available system configurations and the invasiveness of the adaptation actions, which can reduce the usability of the system. Furthermore, adaptation actions perform a system re-configuration. The applicability of user-centric adaptation goes further than the hosted shared DaaS of Section 2. Even if a multi-tenant service instance is not shared by different users (e.g. a VDI DaaS), the underlying infrastructure is still shared. Therefore, user preferences on the service may have an impact on such infrastructural resources, leading to a similar scenario. Figure 1 depicts the activities of the MAPE loop necessary to support user-centric adaptation.

- **Monitoring** has the objective to capture changes in the operational environment. As shown in Figure 2, these can include user-related changes, modifications of available system configurations, and variations of computational resources that can be provided by the underlining infrastructure. Any user, configuration or infrastructure related change can trigger a new adaptation. User-related changes include modifications of the users’ preferences on the available system configurations or variations of the number of users per tenant, which in turn can affect the global preferences of a tenant. Moreover, user-related changes include changes of tenants’ preferences and variations of the tenants that are currently using the system. Monitoring users’ preferences can be performed, for example, by asking for explicit users’ feedback [1] or mining the quality feedback from the users’ behaviour.

- Additionally, 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 configuration and infrastructure changes can have an impact on how the users’ preferences can be satisfied. However, since adaptation can affect the system configuration provided to each tenant, continuous adaptation can negatively affect the usability of the system [11]. For this reason, a new adaptation should only be applied if the previous one was not “too recent”. To achieve this aim, the monitoring activity must also track the last time instant when an adaptation was performed.

- **Analysis** has the objective to identify the best system configuration(s), which optimises a set of metrics. In particular, a candidate re-configuration should maximise the satisfaction of the users’ preferences and minimise adaptation costs, which can be determined by the “intrusiveness” of a reconfiguration (i.e. how negatively a re-configuration will affect the users’ experience). For example, a re-configuration that modifies the look and feel of a DaaS instance is more intrusive than another one that modifies the number of applications that can be opened at the same time. The candidate reconfiguration(s) must also satisfy infrastructure constraints (e.g., maximum storage and computational resources that can be adopted). Analysis of users’ preferences can be - and has been previously [28] - performed by using multi-objective optimisation [20] and constraints solving analysis. However, these techniques have seldom been applied at runtime.

- **Planning** must compare candidate reconfiguration(s) with the current one and determine if adaptation should be performed. In case one of the candidate re-configuration(s) is more suitable than the current one, the outcome of this activity will be an adaptation strategy, indicating how a candidate re-configuration should be applied at runtime. For example, changes in the application look and feel might not be applied until specific users terminate the interaction with the system. A re-configuration that reduces the number of applications that can be executed at the same time, for example to three, can only be applied if a user is running at most three applications at the same time.

- **Execution** has the objective to apply adaptation at runtime. For example, in case of a VDI DaaS model a variant of existing application instances should be deployed dynamically, as proposed in [2, 19, 22, 29, 30]. While, for a hosted shared DaaS model the single application instance should be modified when possible.

### 3.2 Preference-based Analysis for Service Re-configuration

As explained previously, in this paper we focus on the analysis of tenants’ preferences to reconfigure a shared multi-tenant service. It consists of searching for a configuration, in the service configuration space, which satisfies tenants’
preferences in a close-to-optimal way. Elasticity and multi-
tenancy are two characteristics of cloud services that make
them operate under constant changes. Tenants, their pref-

ferences, service infrastructure or even service configuration
space may suffer changes in a dynamic and non-predictable
way, as we described in Subsection 3.1. These changes can
trigger a service reconfiguration, whose goal is to improve
tenants’ satisfaction.

The preferences-based analysis problem is composed of a
four elements tuple

\[ A = \{ C, I, U, F \} \]

where \( C \) represents the configuration space of the service,
\( I \) represents the underlying infrastructure, \( U \) represents the
utility functions of the tenants’ preferences satisfaction and
\( F \) is a function that aggregates the utility of all the tenants
for a specific configuration. \( C, I \) and \( U \) may change at runtime.

Given a time instant \( t \) with a running configuration \( c_{t-1} \),
we define the analysis problem as the search of a \( c_t \) as

\[ F(c_t) \geq F(c_{t-1}) + \delta \]

where \( \delta \) is a constant that measures the cost of applying
the candidate reconfiguration. Tenants’ preferences may
change, i.e. \( U_{t-1} \neq U_t \), and even tenants may leave or join
the service. Therefore, in most of the cases the satisfaction
of all the tenants may decrease \( (F_t(c_{t-1}) < F_{t-1}(c_{t-1})) \).
As a starting point, we consider a simplified version of the
analysis problem

\[ A_t = \{ C, U, F \} \]

where we assume that the infrastructure can support any ser-
vice configuration. In other words, \( C \) and \( I \) are not affected
by any changes, and \( \delta \) is always equal to 0. Then the only
triggers of the analysis are changes in the tenants and/or their preferences.

4. SOLUTION

This section presents our approach for the preference-
based analysis. As shown in Figure 3, we consider potential

![Configuration space](Image)

Figure 2: Changes in the operational environment captured by the monitoring activity.

service configurations (i.e. configuration space), users’ pref-

erences expressed on the configurable service elements, and
the current configuration of the service as inputs of our anal-
ysis. The configuration space is expressed as an EFM (Sub-
section 4.1), and the tenants’ preferences are represented by
using a preference model (Subsection 4.2). We interpret
the analysis as a coalitional game of game theory (Subsec-
tion 4.3), which is solved conceptually by the Nash Bargain-
ing Solution [25] and practically by using multi-objective
optimization.

For this paper we made the assumption that every ten-
ant groups different users who share common preferences.
The clustering of the users’ preferences into different ten-
ants is performed by an external process, which is out of the
scope of this work. The preferences are obtained during the
monitoring phase and are translated into preferences terms,
described in Section 4.2.

4.1 Configuration Space

The configuration space of the DaaS scenario is repres-
ented as an Extended Feature Model (EFM). An EFM rep-
resents the configuration space of a software system in terms
of functional features and non-functional attributes. As
shown in Figure 4, available configuration options associated
with our DaaS example are Look & Feel, Security, Mainte-
nance, and Updates.

Functional features are graphically represented as nodes
and are connected among them hierarchically. They can
simply be selected or not. Examples of functional features
are Aero and Classic for Look & Feel. Feature relationships
are individual - mandatory (black circle) and optional (white
circle) - or grouped. For grouped relationships, a cardinality
determines how many children features can be selected. For
example, Security is mandatory, while Firewall, Encryption,
and Antivirus are optional features. Look & Feel presents a
grouped relationship with a \{1,1\} cardinality (i.e. only one
sub-feature among Aero and Classic can be selected). While
for application updates (AppUpdate) at least one feature
among eclipse, office, and java updates should be selected.

Non-functional attributes are formally represented as vari-
ables with a domain. An attribute is linked to a feature, and
optionally to other attributes by means of constraints. An
example of a feature attribute can be the security level of
the firewall that can assume four possible values (Low, Medium,
High, and Complete). The constraint shown in Figure 4 im-

![Preference-based Analysis](Image)

Figure 3: Inputs and outputs of the Preference-based Analysis.
preferences that, when the firewall security level is Complete, feature application updates (AppUpdates) should be disabled.

We decided to represent the configuration space as an EFM for several reasons. First, EFMs - in particular - and Variability Models - in general - have been extensively used to represent variability-intensive systems for academic and industrial purposes [5]. Second, an EFM can express a large configuration space in a compact way, which is also amenable to a graphical representation. For instance, if we consider both features and attributes, the model shown in Figure 4 represents 1,053,184 different configurations.

Finally, a wide catalog of analysis operations [3] is also available to extract information from these models and, therefore, we can apply or tailor existing analysis operations to our purposes.

### 4.2 Tenants’ Preferences

To express tenants’ preferences we assume that the tenants agree on the hard requirements of the service. For example, in our case they agree on the operating system (Windows) and the delivery model of the DaaS (hosted shared). However, each tenant can express different preferences on the configurable service elements (i.e., features and attributes shown in Figure 4). A service cannot satisfy conflicting hard requirements, but can provide a balance between conflicting soft preferences.

We adapt some of the preference terms of Semantic Ontology of User Preferences (SOUP) preference model [12] to express the tenants’ preferences. SOUP is “a highly expressive, intuitive model of user preferences”. Initially, it was designed to express preferences on service discovery and ranking [13]. However, it has been adapted to different scenarios, such as resources allocation in business processes [7]. SOUP defines preferences terms which refer to functional and non-functional terms. In our case, functional terms are features and non-functional terms are attributes. In particular, we are interested in five preferences terms:

- **Favorites preference term**: A favorites preference defines a finite set of values that constitute the desired values of the referred element. For our case, a favorites preferences refers to a feature we want to be selected. For example, we may favorite feature *Aero*.

- **Dislikes preference term**: A dislikes preference defines a set of property values that the service should not provide for the referred element. For our case, a dislikes preference refers to a feature we want to be removed. For example, we may dislike feature *JavaUpdt*.

- **Highest preference term**: A highest preference prefers values as high as possible for the referred element. For our case, a highest preference refers to an attribute whose value we want to maximise. For example, we may want the highest value for attribute *Firewall.level*.

- **Lowest preference term**: A lowest preference prefers values as low as possible for the referred element. For our case, a lowest preference refers to an attribute whose value we want to minimise. For example, we may want the lowest value for attribute *JavaUpdt.period*.

- **Around preference term**: An around preference determines which value is better by determining the distance of each value to a concrete value provided as an operand of this preference term. For our case, an

<table>
<thead>
<tr>
<th>Type</th>
<th>Preference</th>
<th>Element</th>
<th>Mapping</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qualitative</td>
<td>Favorites(f)</td>
<td>Feature</td>
<td>( f = \text{selected} \implies p_{ij} = 1 )</td>
<td>Favorites(Aero)</td>
</tr>
<tr>
<td>Qualitative</td>
<td>Dislikes(f)</td>
<td>Feature</td>
<td>( f = \text{removed} \implies p_{ij} = 1 )</td>
<td>Dislikes(JavaUpdt)</td>
</tr>
<tr>
<td>Quantitative</td>
<td>Highest(att)</td>
<td>Attribute</td>
<td>( p_{ij} = \text{upperBound} - \text{lowerBound} )</td>
<td>Highest(Firewall.level)</td>
</tr>
<tr>
<td>Quantitative</td>
<td>Lowest(att)</td>
<td>Attribute</td>
<td>( p_{ij} = \text{upperBound} - \text{lowerBound} )</td>
<td>Lowest(JavaUpdt.period)</td>
</tr>
<tr>
<td>Quantitative</td>
<td>Around(att,d)</td>
<td>Attribute</td>
<td>( p_{ij} = \text{inverseDistance} (\text{value}, d) )</td>
<td>Around(Antivirus.frequency, 3)</td>
</tr>
</tbody>
</table>

Figure 4: DaaS configuration space expressed as an Extended Feature Model

Table 2: Adapted SOUP preferences and mapping.
around preference defines which value we want an attribute to be close to. For example, we may want attribute Antivirus frequency to be close to 3 days.

Table 2 shows the five aforementioned preference terms, together with examples and their quantification to measure tenants' satisfaction. We have adapted SOUP preferences to work with features and attributes. Initially, described preference terms were intended to define a partial ranking between a set of items. However, we have decided to compute a satisfaction value (i.e. a real number between a set of items). However, we have decided to compare different implementations of our analysis in order to find the best one. Since features have a boolean domain, we use both Favorites and Dislikes preferences to express features selection. Highest, Lowest and Around preferences are associated with attributes, although their domain should be ordered. For Highest and Lowest preferences, the closer the value of an attribute is to the highest or the lowest target value, respectively, the closer to 1 the value of \( p \) will be. Around is the only preference that needs an extra operand, which defines a desired value. The lower the distance of the attribute value is to such desired value, the closer to 1 the value of \( p \) will be.

### 4.3 Analysis

The goal of the analysis is to obtain the configuration that best fits the different tenants' preferences. Figure 3 shows the inputs of the analysis: the service adaptation space expressed as an EFM, the tenants' preferences and the current configuration of the service. The output is a reconfiguration of the service that should satisfy current tenants' preferences better than the previous configuration.

We interpret this analysis as a problem of game theory. Game theory is a discipline which deals with "the study of mathematical models of conflict and cooperation between intelligent rational decision-makers" [23]. A game is defined as a three-elements tuple, \( \Gamma = (N, C, U) \), \( \forall i \in N \) where we have different players \( N \) who use several strategies \( c_i \), \( \forall i \in N \) to maximise their fitness function \( u_i \), \( \forall i \in N \). Games can be cooperative or non-cooperative, depending if the players can form coalitions to accomplish better results than in an independent way. We have considered a cooperative game since we can achieve the maximum overall satisfaction of tenants' preferences. Coalitions can be ruled by communication among players, or by means of an impartial arbitrator. The solution that maximises players' fitness functions is the Nash Bargaining Solution [24, 23], which is a Pareto efficient solution maximising the Nash Product \( \prod_i u_i \). We consider that all the tenants make a grand coalition and that our analysis acts as the impartial arbitrator. In particular, our analysis obtains the service configuration (Nash Bargaining Solution) that best fits tenants' preferences \( U \) over the service adaptation space \( C \).

Since game theory is a general discipline, we use specific techniques (i.e. multi-objective optimisation techniques) to compute a solution. We translate the tenants' preferences to fitness functions using the mapping shown in Table 2. A fitness function is expressed as \( u_i = \sum_j p_{ij} \). In this way, we can measure the satisfaction of a single tenant as the sum of the satisfaction of its preferences. Each fitness function represents a different objective of the multi-objective optimisation problem. For a non-trivial multi-objective optimisation problem, it does not exist a single solution that simultaneously optimises each objective. In this case, the objective functions are said to be conflicting, and there exists a (possibly infinite) number of Pareto optimal solutions. To choose a single solution, we apply a weighted Nash Product \( \prod_i u_i w_i \), where \( w_i \) defines the weight or importance of each tenant, in order to balance the satisfaction of the different tenants. We consider specifically a Normalized Nash Product to compare different solutions from the Pareto front.

\[
\text{NNP} = \prod_i \frac{u_i w_i}{U_{\text{MAX}}} \frac{U_{\text{MAX}}}{U_{\text{MAX}}}
\]

This NNP is the aggregation function \( F \) defined in Subsection 3.2. \( U_{\text{MAX}} \) and \( U_{\text{MAX}} \) are the maximum possible value of \( u_i \) for a specific tenant and the value of \( u_j \) of all the tenants, respectively. The analysis returns the solution that maximises this product.

Table 3 shows an analysis scenario based on the DaaS example of Section 2, where \( w_i \) is given by the number of current users of tenant \( i \). In the first snapshot we have 3 tenants, and the DaaS is running a configuration \( c_i \) which provides the satisfaction \( u_i \) of each tenant. Note that at this stage the satisfaction of tenant \( i_3 \) is very low because its preferences have not yet been taken into account during the analysis. In the second snapshot we have several changes: the preferences of tenants \( i_2 \) and \( i_3 \) vary, there is a new tenant \( i_4 \), and the number of users of every tenant also change. Consequently, the utility value of the current configuration \( (u_i(c_i)) \) changes accordingly, becoming sub-optimal. Indeed the preference-based analysis is re-executed and returns a new configuration, \( c_j \), which delivers optimal utility values \( u_i(c_j) \). The most remarkable improvement is for tenant \( i_4 \), whose preferences' satisfaction increases from 0.5 to 3.5. The improvement of the global satisfaction of tenants' preferences is also indicated by the NNP (from 1.12 to 5.76).

### 5. EVALUATION

This section illustrates the implementation of the prototype we adopted to perform the preferences analysis. It also describes how we conducted our experiments and discusses obtained results.

#### 5.1 Implementation

We implemented a prototype to perform and evaluate the proposed users' preferences analysis. We used jMetal - a java based metaheuristics framework for multi-objective optimisation problems [10]. jMetal provides a number of algorithms to solve multi-objective optimisation problems (i.e. to compute a Pareto front of the problem). Among all the algorithms that jMetal provides, we have chosen a genetic algorithm, specifically FastPGA, due to its wide use for the analysis of EFMs [16, 28]. Since we are interested in the solution that balances and maximises tenants' satisfaction, we look to maximise the NNP defined in Section 4.3. In case at least one tenant has a satisfaction value equal to 0, NNP returns 0. If all the returned points of the Pareto front have a NNP = 0, then we select a solution that maximises the average satisfaction of the tenants' preferences, where the satisfaction of each tenant is weighted according to its importance (weight \( w \)).
Table 3: Preferences reconfiguration scenario. Underlined preferences highlight changes, while blank lines represent removed preferences.

<table>
<thead>
<tr>
<th>Preferences</th>
<th>Snapshot1</th>
<th>Preferences</th>
<th>Snapshot2</th>
</tr>
</thead>
<tbody>
<tr>
<td>tenant1</td>
<td></td>
<td>- Favorites(OfficeUpdt)</td>
<td></td>
</tr>
<tr>
<td>- Favorites(Aero)</td>
<td>45</td>
<td>- Highest(Firewall.level)</td>
<td>49</td>
</tr>
<tr>
<td>- Around(Backup.period,Weekly)</td>
<td>3</td>
<td>- Around(OfficeUpdt.period,Weekly)</td>
<td>3</td>
</tr>
<tr>
<td>- Lowest(Antivirus.frequency)</td>
<td></td>
<td>- Highest(Antivirus.frequency)</td>
<td>3.5</td>
</tr>
<tr>
<td>- Around(Firewall.level,Medium)</td>
<td></td>
<td>- Highest(Firewall.level)</td>
<td></td>
</tr>
<tr>
<td>tenant2</td>
<td>60</td>
<td>- Lowest(Firewall.level)</td>
<td>53</td>
</tr>
<tr>
<td>- Likes(Indexing)</td>
<td></td>
<td>- Around(Firewall.level,Medium)</td>
<td>3</td>
</tr>
<tr>
<td>- Likes(Defragmenter)</td>
<td>2.5</td>
<td>- Likes(Indexing)</td>
<td></td>
</tr>
<tr>
<td>- Dislikes(Classic)</td>
<td></td>
<td>- Likes(Defragmenter)</td>
<td></td>
</tr>
<tr>
<td>tenant3</td>
<td>31</td>
<td>- Around(OfficeUpdt.period,Weekly)</td>
<td>40</td>
</tr>
<tr>
<td>- Highest(Antivirus.frequency)</td>
<td></td>
<td>- Highest(OfficeUpdt.period,Weekly)</td>
<td>4</td>
</tr>
<tr>
<td>- Highest(Backup.period,Weekly)</td>
<td></td>
<td>- Highest(Antivirus.frequency)</td>
<td>3.5</td>
</tr>
<tr>
<td>- Highest(Firewall.level)</td>
<td></td>
<td>- Highest(Defragmenter)</td>
<td></td>
</tr>
<tr>
<td>tenant4</td>
<td>0</td>
<td>- Around(JavaUpdt.period,Monthly)</td>
<td>23</td>
</tr>
<tr>
<td>- Highest(JavaUpdt.period,Monthly)</td>
<td></td>
<td>- Around(Backup.period,Weekly)</td>
<td>0.5</td>
</tr>
<tr>
<td>- Highest(Firewall.level)</td>
<td></td>
<td>- Around(Antivirus.frequency,3)</td>
<td></td>
</tr>
<tr>
<td>- Highest(Firewall.level)</td>
<td></td>
<td>- Highest(Firewall.level)</td>
<td>3.5</td>
</tr>
</tbody>
</table>

**Normalized Nash Product (10^8)**

Table 4: Amount of changes between two consecutive snapshots t−1 and t.

<table>
<thead>
<tr>
<th>t−1</th>
<th>t−1</th>
<th>t−1</th>
<th>t−1</th>
</tr>
</thead>
<tbody>
<tr>
<td>card(T)</td>
<td>n</td>
<td>∈ {n−1, n+1}</td>
<td></td>
</tr>
<tr>
<td>card(Pi)</td>
<td>mi</td>
<td>∈ {mi−1, mi, mi+1}</td>
<td></td>
</tr>
<tr>
<td>wij</td>
<td>wij</td>
<td>∈ [WMIN, WMAX]</td>
<td></td>
</tr>
</tbody>
</table>

5.2 Experiments

We present a preliminary experimental study to check the feasibility of our approach. In this study, we consider scenarios where tenants are joining and leaving, and their preferences and number of users are evolving between different snapshots. For every snapshot, we run an analysis to reconfigure the service. We compare the satisfaction achieved by each reconfiguration to the satisfaction value obtained for the previous analysis. The result is also compared to the satisfaction of a random service configuration.

For our experiments, we define T as a set of tenants, where each one has a number of users wij and a set of preferences Pi. wij is defined in the integer range [WMIN, WMAX], considering also that ∑ wij ≤ WTOTAL. The number of tenants card(T) is defined in the integer range [TMIN, TMAX], and the number of preferences per tenant card(Pi) is defined in the integer range [PMIN, PMAX].

Table 4 shows the amount of changes between two consecutive snapshots. For each snapshot 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, it can be associated with a new preference or an old preference is removed. The weight of every tenant may vary between the maximum and minimum values. We also performed experiments by using different configuration spaces. To achieve this aim, we generated four additional EFMs by using BeTTy [31], a well-known EFM generator. Table 5 shows the characteristics of the considered EFMs. CTC means the number of Cross Tree Constraints (non-hierarchical constraints) of each model.

For each EFM model shown in Table 5, we executed 25 different tenant changes scenarios. We randomised the num-
Table 5: Characteristics of the EFMs used for our experiments.

<table>
<thead>
<tr>
<th></th>
<th>Configs</th>
<th>Feats.</th>
<th>Atts.</th>
<th>CTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daas</td>
<td>1 053 184</td>
<td>18</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>BeTTy1</td>
<td>49 483 952</td>
<td>20</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>BeTTy2</td>
<td>1.01 \times 10^9</td>
<td>20</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>BeTTy3</td>
<td>&gt; 2.15 \times 10^9</td>
<td>30</td>
<td>21</td>
<td>9</td>
</tr>
<tr>
<td>BeTTy4</td>
<td>&gt; 2.15 \times 10^9</td>
<td>30</td>
<td>18</td>
<td>9</td>
</tr>
</tbody>
</table>

The number of snapshots per scenario \( n \) in the integer range \([5, 10]\). Initial values and ranges for the remaining parameters are as follows: \( T_{\text{min}} = 2, T_{\text{max}} = 5, P_{\text{min}} = 2, P_{\text{max}} = 10, W_{\text{MIN}} = 10 \) and \( W_{\text{MAX}} = 80 \). Since each different tenant implies a new objective, we select the same upper limit \( (T_{\text{max}} = 5) \) chosen in related papers on multi-objective optimisation for Feature Models (FMs) [28, 27]. We consider \( W_{\text{TOTAL}} = 200 \), since such value is close to the maximum number of users supported by a single real hosted shared Daas\(^7\). For the FastPGA algorithm we configured the parameters provided by \( jMetal \) as follows: \( \text{Evaluations} = 25000, \text{PopulationSize} = 100, \text{CrossoverProbability} = 0.9, \) and \( \text{MutationProbability} = 0.05 \).

5.3 Results and Discussion

Table 6 shows the average execution time and average satisfaction of tenants’ preferences obtained for our experiments. Execution time increases in a linear way, since we are using the same number of FastPGA evaluations for all the EFMs. However, the larger the EFM, the lower the average satisfaction achieved, because the number of evaluations remains constant. This highlights that we need to adapt the number of evaluations depending on the size of the model. We also noticed that the number of tenants does not produce any significant impact on the average execution time and tenants’ satisfaction.

Table 6: Results of the preference-based analysis experimental study.

<table>
<thead>
<tr>
<th></th>
<th>Time (ms)</th>
<th>Satis.</th>
<th>( C_t &gt; C_R )</th>
<th>( C_t &gt; C_{t-1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daas</td>
<td>8 497</td>
<td>72.79%</td>
<td>98.88%</td>
<td>63.48%</td>
</tr>
<tr>
<td>BeTTy1</td>
<td>12 225</td>
<td>64.12%</td>
<td>88.33%</td>
<td>53.33%</td>
</tr>
<tr>
<td>BeTTy2</td>
<td>13 701</td>
<td>59.76%</td>
<td>66.67%</td>
<td>38.33%</td>
</tr>
<tr>
<td>BeTTy3</td>
<td>18 721</td>
<td>59.99%</td>
<td>83.89%</td>
<td>48.89%</td>
</tr>
<tr>
<td>BeTTy4</td>
<td>18 400</td>
<td>46.87%</td>
<td>73.52%</td>
<td>30.59%</td>
</tr>
</tbody>
</table>

\( C_t > C_R \) measures the percentage of executions identifying a configuration whose satisfaction is better than a valid configuration randomly generated \( (C_R) \). For our Daas example, this measure reaches 98.88%. Although it decreases for the rest of the models, we estimate that the reason of such decrease is also the constant number of evaluations. Figure 5 shows the average value of such measure, going from 20% to 40% depending on the model.

Finally, \( C_t > C_{t-1} \) measures the percentage of executions that finds a reconfiguration that improves the tenants’ preferences satisfaction compared to that obtained with the current configuration. We can see how this measure generally decreases for larger models. However, such results are affected by the “amount of change”, i.e. the distance in terms of tenants and preferences between two consecutive snapshots. The larger the amount of change, the larger the subset of possible better reconfigurations. Therefore for a larger amount of changes we expect an improvement of the satisfaction of the tenants’ preferences between two consecutive snapshots. However, since our prototype is only able to handle a fixed number of evaluations the average improvements in the satisfaction of the tenants’ preferences between two consecutive snapshots is only around 7 – 10%.

This is a first experimental study. In order to arrive to stronger conclusions, we need to extend this study in several ways:

1. Extend our current generator of changes scenarios to explicitly measure the amount of changes produced.
2. Try a different criteria to terminate the analysis. For example we can adapt the maximum number of evaluations depending on the size of the EFM, or consider a time limit instead.
3. Compare different \( jMetal \) algorithms and parameters. Currently we tried with FastPGA algorithm. However, depending on the amount of change or the model size, different algorithms or parameters may work better.

6. RELATED WORK

Users’ preferences have been recently considered as a main adaptation trigger. In particular, social adaptation [1] proposes to dynamically adapt existing software systems depending on the users’ collective judgement on the way the system should behave. This approach treats users’ feedback as a primary driver for planning and guiding adaptation. However, they consider different kinds of feedback, while we focus on the users’ preferences. The scope of the decision making is also different. Our analysis searches a specific configuration among the whole system configuration space, while their analysis decides between the existing alternatives in order to fulfill a specific requirement. Song et al. [32] present an approach to develop self-adaptive systems that take into account end-users, who express their preferences redressing the adaptation result. In our approach, we consider an explicit preference model. Dapiz et al. [9] focus on user preferences over non-functional properties as a key driver for adaptation. However, this approach has its main
focus on pervasive infrastructures and describes preferences through user routines.

Cloud services analysis and adaptation has been a prolific research area during the last years. Caton and Rana [8] 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. [21] also propose an adaptive resource configuration for cloud infrastructure management. In this case, they structure adaptation actions into levels, rely on Case-Based Reasoning and a rule-based approach, and even consider SLA-violations. Wei et al. [34] present a similar idea, with the difference that they use a game theoretic method based on Nash equilibria. They intend to reach an equilibrium in the different resource allocations. Inzinger et al. [17] focus on cloud applications instead of the infrastructure. They propose a model-based adaptation which allows users to specify application behavior, and providers to consider data from multiple customers to offer better adaptation decisions.

Research on Software Product Lines (SPLs) is highly related to our paper. The idea of using variability techniques to model the adaptation space is not new. For example, Bencomo et al. [4] propose the use of variability modelling to define the runtime adaptation space. About multi-tenancy and SPLs, Schroeter et al. [29, 30] use variability and SPLs to assist the configuration of multi-tenant applications. The authors identify configuration requirements and propose a configuration process using EFM. Mietzner et al. [22] propose to use variability modelling techniques to manage the variability of Software as a Service (SaaS) applications and their requirements. Specifically, they use variability models to configure and deploy SaaS applications for different tenants. However, they focus on modelling the variability and deploying different variants of a SaaS application instance. Variability of different cloud providers has also been analysed and modelled by García-Galán et al. [14], in order to assist the migration of an in-house infrastructure to the cloud. However, this approach works with hard requirements and ignores changes of users’ preferences.

Several research efforts have been made to investigate multi-objective optimisation in applications characterised by variability. Guo et al. [16] use a Genetic Algorithm to find optimal FM configurations for a single objective and user. Sayyad et al. [28, 27] perform multi-objective optimisation of several large EFM using metaheuristics techniques. However, their objective functions are fixed (minimise errors and cost, or maximise number of features), while our fitness function depends on the specific users’ preferences. Finally, other work has explored techniques for solving conflicts in a configuration process. White et al. [35] propose a technique in this direction that only considers a single user and a minimal changes criterion. While García-Galán et al. [15] consider multiple users, but after detecting the conflicts, the users have to define explicitly the impact of every solution on their preferences’ satisfaction.

7. CONCLUSIONS

In this paper, we have presented an approach to support user-centric adaptation of multi-tenant services. We have motivated user-centric adaptation by using a DaaS example, and we have explained how to engineer the activities of the MAPE loop necessary to support it. Although our intention was to present an approach valid for any kind of adaptive system, we focused our attention on multi-tenant services and on the preference-based analysis for identifying the service configuration that maximise tenants’ satisfaction. The provided solution takes inspiration from coalition game theory. We have expressed tenants’ preferences by using SOUP preference model, while the configuration space of the service is represented as an EFM. Obtained experimental results demonstrate that our analysis approach effectively maximises tenants’ preferences and can be performed at runtime. However, our prototype did not exhibit good results for improving tenants’ satisfaction with very large configuration spaces. This means that the time required to perform the analysis can increase with the number of configurations, and, for this reason, a different analysis technique may be required for larger configuration spaces.

As future work, firstly we propose to extend our experiments to better understand the impact of the configuration space, tenants and their evolution on the analysis results. In particular, we can compare different heuristics and algorithms to detect which one is more suitable for each specific case. Modelling more realistic multi-tenant scenarios including a larger number of configurations will also be necessary. Moreover we will extend our analysis, by including constraints in the configuration space and in the computational resources that can be provided by the infrastructure. The cost of applying a candidate reconfiguration will also be taken into account to select the best one. Finally, regarding the whole user-centric adaptation problem, we will address the rest of the activities of the MAPE loop. Indeed we will investigate how to monitor users’ preferences in a non-intrusive way and how to apply a reconfiguration on the system at runtime. We plan to perform tests with users in order to determine time thresholds for specific adaptation actions.

8. ACKNOWLEDGEMENTS

Special thanks to J.A. Parejo for his advices for the experimental study, to J.M. Garcia for his explanations about preferences modelling, and to A. Sayyad for his useful comments about multi-objective optimisation in FMs.

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 SFI grant 10/CE/I1855, and the ERC Advanced Grant (ASAP) no. 291652.

9. BIBLIOGRAPHY


