Automating the Support of Highly-configurable Services

Jesús García Galán

PhD dissertation

Supervised by

Dr. Antonio Ruiz Cortés

and

Dr. Pablo Trinidad Martín-Arroyo

Universidad de Sevilla

June 2015

First published in May 2015 by
Jesús García Galán

http://www.isa.us.es/members/jesus.garcia
jegalan@us.es

Support: PhD dissertation granted by the University of Seville, the European Commission (FEDER) and Spanish Government under CICYT project SETI (TIN2009-07366) and TAPAS (TIN2012-32273), and the Andalusian Government projects COPAS (P12-TIC-1867) and THEOS (TIC-5906)

A mi familia

AGRADECIMIENTOS

Bernardo de Chartres dijo una vez que “si he logrado ver más lejos, ha sido porque he subido a hombros de gigantes”. Aunque esta cita se utiliza con frecuencia en el contexto científico, no es menos real en la vida diaria. Ninguno de nosotros sería lo que somos hoy día sin la ayuda de esos “gigantes” que tenemos alrededor.

En primer lugar, quiero acordarme de los magníficos compañeros de los que he disfrutado estos años, ya casi 7 desde que comencé a trabajar en la Universidad de Sevilla. Toda esta aventura comenzó incluso antes, en el aõ 2007, cuando conocí a Pablo Trinidad y Carlos Muller como profesores en la asignatura ISG2. Poco después, tras terminar el proyecto de la ingeniería técnica con Pablo, empecé a trabajar para el grupo ISA como desarrollador. En ese periodo tuve la suerte de conocer a grandes compañeros y amigos como José Galindo, Ana Belén o Guti entre otros, con los que he compartido un camino casi paralelo y he vivido muy buenos ratos profesionales y personales. Poco a poco también tuve la ocasión de conocer a esos que antes habían sido mis profesores, y de los que hoy día sigo aprendiendo como compañeros: Sergio, Carlos, Jose, Jose Mari, David... la lista es larga, siento si me olvido algún nombre. Tampoco quiero olvidarme de los amigos que fui conociendo durante la carrera, y de Andrés, mi compañero de despacho, con el que aún queda pendiente un partido de futbol.

También debo dar especialmente las gracias a aquellos con los que he trabajado mas de cerca durante el doctorado. Y debo empezar por Pablo y Antonio, mis tutores de tesis. Gracias por todo lo que he aprendido de vosotros, por el tiempo y el esfuerzo dedicado, por la confianza puesta en mí y por las posibilidades que también habéis puesto en mis manos. Esta tesis no hubiera sido la misma sin vosotros. Como tampoco lo hubiera sido sin las estancias que he disfrutado en Valencia, Cardiff y Limerick. Gracias a Vicente Pelechano, Omer Rana y Mike Hinchey por brindarme esas oportunidades, y a toda la gente conocí en la Universidad Politécnica de Valencia (y concretamente en el grupo PROS: María, Miriam, Mario, Pablo, Joan...), en la Universidad de Cardiff y en Lero. Gracias especialmente a Omer, con el que desde entonces mantengo fructíferas colaboraciones, y cuya idea de analizar los servicios de Amazon es una de las principales patas de esta tesis; y a Liliana, no solo como una excelente

compañera profesional sino tambien como una gran amiga con la que he pasado ratos realmente divertidos.

Y finalmente, quiero acordarme de mi familia. De esa familia que uno va haciendo a medida que va avanzando en la vida. Gracias a todos esos amigos que han estado dispuestos a darme un apoyo en estos años, con los que he compartido tantas buenas experiencias, y con los que espero seguir compartiéndolas. Especialmente gracias a Ale, a Sergio, a Alejandro, a Jose y a Tagua, y también a tantos otros con los que las paginas de esta tesis podrían aumentar considerablemente. Y por supuesto, gracias a mi madre, por inculcarme su espíritu de lucha y superación, y por haber puesto todo su esfuerzo en que pueda valerme por mí mismo.

De todo corazón, gracias.

Jesús.

Resumen

Las crecientes capacidades de configuración de los servicios, especialmente en el cloud, han dado lugar a los así llamados Servicios Altamente Configurables (HCSs). Dichas capacidades de configuración están aumentando la demanda y complejidad de las aplicaciones basadas en HCS y de las necesidades de infraestructura para soportarlas, soluciones guiadas por HCS de aquí en adelante. Tras un estudio del estado del arte, concluimos que dichas soluciones guiadas por HCSs pueden ser mejoradas significativamente en 1) los lenguajes para describir las configuraciones, también conocidas como el espacio de decisión del servicio, y 2) en las técnicas para extraer información de utilidad del espacio de decisión, técnicas de análisis en adelante.

Por un lado, no existen Lenguajes Específicos de Dominio (DSLs) para describir el espacio de decisión, aunque hay algunas aproximaciones cercanas. En esta tesis creemos que es posible mejorar el actual panorama diseñando un DSL: i) en conformidad con los principales proveedores de HCSs, ii) que soporte múltiples ítems, iii) suficientemente expresivo para facilitar la descripción de relaciones lógicas y aritméticas en los términos del servicio y iv) independiente del dominio. Adicionalmente, este DSL debe definir criterios de validez para asegurar que el espacio de decisión satisface propiedades básidas como la consistencia y la configurabilidad. Más allá, se deben ofrecen explicaciones cuando el espacio de decisión del servicio no satisfaga dichas propiedades.

Por otro lado, la mayor parte de las actuales técnicas de análisis, como aquellas encargadas de encontrar configuraciones óptimas o de reconfigurar servicios multitenant, llevan consigo algunos inconvenientes propios de técnicas emergentes. Para superar estos inconvenientes, se debe desarrollar: a) implementaciones de referencia totalmente funcionales, b) técnicas reusables, c) mecanismos de extensión efectivos y d) interfaces amigables.

El principal objetivo de esta disertación es mejorar el soporte existente para el desarrollo de soluciones guiadas por HCS, considerando las recomendaciones anteriores. Las principales contribuciones de esta tesis son un DSL (llamado SYNOPSIS) para especificar el espacio de decisión de los HCS, y un nuevo catálogo de operaciones para

comprobar y explicar los criterios de validez así como para encontrar configuraciones óptimas para uno o más usuarios. Como contribuciones menores, también presentamos dos soluciones que han sido desarrolladas para mejorar el existente soporte para la migración de infraestructuras a Amazon EC2 y para reconfigurar servicios multi-tenant.

La piedra angular de nuestra propuesta para mejorar las técnicas de especificación ha sido definir un DSL, SYNOPSIS, y dotarlo con semántica formal basada en Modelos de Características con Estados. Acerca de nuestra propuesta para mejorar las técnicas de análisis, la clave ha sido la organización de dichas técnicas en un catálogo de operaciones básicas que pueden ser combinadas para dar lugar a soluciones guiaadas por HCS más avanzadas. La aplicabilidad de nuestros resultados está limitada a aquellos espacios de decisión que pueden ser transformados en Modelos de Características con Estados, que por nuestra experiencia es suficiente para soportar HCSs reales.

Abstract

The growing customisation capabilities of services, especially in Cloud scenarios, have led to the so-named Highly-configurable Services (HCSs). Such capabilities are boosting the demand and complexity of HCS-based applications and the infrastructure need to support them, HCS-driven solutions from now on. After a study of the existing literature we conclude that these HCS-driven solutions can be significantly enhanced in both 1) the languages to describe the configurations, a.k.a. the decision space of HCSs and 2) the techniques to extract useful information from the decision space, analysis techniques, in advance.

On the one hand, there are no Domain Specific Languages (DSLs) to describe the decision space, although there exist some very close approaches. We suggest that it is possible to improve the current landscape by designing a DSL: i) Compliant with big HCS vendors, ii) multi-item aware, iii) expressive-enough to ease the description of arithmetic-logical relationships on configurable description terms, and iv) domain-independent. In addition, this DSL must define validity criteria for checking that the decision space satisfies some basic properties such as the consistency, and the configurability. Furthermore, explanations must be provided when the decision space do not satisfy such basic properties.

On the other hand, most of the current analysis techniques such as those relate to find optimal configurations or reconfigure a multi-tenant service includes some drawbacks that can be found in emerging techniques. To overcome such drawbacks there must be developed: a) fully-functional reference implementations, b) techniques with a reuse-oriented design, c) effective extension mechanisms, and d) user-friendly interfaces.

The main goal of this dissertation is to enhance the existing support to develop HCS-driven solutions considering the aforementioned suggestions for improvement. The main thesis contributions are a DSL to specify the decision space of HCSs called SYNOPSIS, and a novel catalogue of analysis operations to check and explain validity criteria as well as to find optimal configurations for one or many users. As minor contributions, two solutions have been developed to improve the existing tooling support

to migrate on-premise infrastructure to Amazon EC2 and to reconfigure multi-tenant service.

The cornerstone of our proposal to improve the specification techniques has been to define a DSL, SYNOPSIS, and endow it with a formal semantics based on Stateful Feature Models. Regarding our proposal to improve the analysis techniques, the key has been the organization of such techniques in a catalogue of basic analysis operations that can be combined to support more advanced HCS-driven solutions. The applicability of our results is limited to those decision spaces that can be translated to a Stateful Feature Model, that is enough to support real-world HCSs, in our experience.

Contents

List of Figures

List of Tables

Part I

Introduction

Introduction

"Which road do I take?" Alicia asked. "Where do you want to go?" responded the Cheshire Cat.

Alice in Wonderland (1865),
Lewis Caroll

In this chapter we introduce and motivate this dissertation. Section §1.1 presents the research context. Section §1.2 briefly describes the goals of the dissertation, while contributions are summarised in Section §1.3. Section §1.4 describes the context of the thesis, and finally Section §1.5 shows the structure of the rest of the document.

1.1 Research Context

During the last years, the use of services has been generalised thanks to cloud computing [104], which has provided the necessary momentum to make software services a major business [8, 27]. Up to date, there exist a plethora of cloud providers that offer services that go from an infrastructure level – such computing or storage services – to end-user software – such as music streaming or office apps. Most of these services are *configurable services*, i.e. services where one or more terms present different alternatives to be chosen by the consumer.

The goal of configurable services is to the adaptation of services to the specific needs of their consumers, covering wider market segments, and consequently increasing the number of potential clients. An example of a configurable service is Dropbox [50], which can be contracted on different plans. Each plan determines specific values for the storage limit and the file sharing options. While the basic plan is free, offering a storage limit of 2 GB and simple file sharing options; the pro plan costs 10 dollars per month, but provides a 1000 GB limit and advanced file sharing control.

The advanced configuration capabilities of some of these configurable services lead us to employ the term *Highly-configurable Service (HCS)*. A HCS can be intuitively defined as a service whose configuration support is not properly handled by the current technology. Besides decision terms, HCSs enable the configuration of multiple service items and may be even related to additional services. This is the case of Amazon EC2 [5], which enables the hiring of multiple customisable computing instances and additional instance storage through the *Elastic Block Store (EBS)* service.

Although the configuration capabilities of HCSs suppose a clear advantage to the service clients, such capabilities may also hinder the decision making. The selection of the best configuration may not be a trivial task, especially if performed manually. First, it is necessary to fully understand the different service configurations, a.k.a. the service decision space, and then to navigate through the different decision terms to discard the values which do not satisfy the required requirements. Finally, we have to evaluate the remaining configurations in order to choose the most suitable. Although for a service like Dropbox this decision making may seem trivial, for services such as Amazon EC2, which almost 17'000 configurations and thousands of dependencies [73], it is clearly not.

As the increasing use of commercial HCSs in which the wrong configuration and

selection may imply an economic lost for clients and providers, the complexity and demand of systems supporting the HCS lifecycle also increase. We refer to such systems as *HCS-driven solutions* from now on. Our study of the different approaches reveals that there is a lack of specific techniques to develop HCS-driven solutions, especially for the specification languages; the techniques to extract useful information from the HCSs, *analysis techniques* from now on; and the tooling support.

In this dissertation we focus on some issues related to the specification, the analysis and the tooling support for HCSs. Regarding specification, as far as we know there is no specific proposals to describe the configuration capabilities [1] of HCSs. In this sense, some general approaches have been employed – with drawbacks – to describe the decision terms of services in real-world scenarios:

- Commercial service providers, such as Amazon [4], Dropbox [50] or Rackspace [129] describe the configuration capabilities of their services for marketing or legal purposes. In order to examine the whole decision space, the potential clients have to navigate through multiple documentation pages which describe the terms and their values in heterogeneous ways. While some values and terms are structured in tables, some others are presented in natural language at the end of large descriptions. This kind of description not only hardens the examination of the service, but also the automated processing required to enable analysis techniques and tooling support.

- Approaches from the services field, such as WS-Agreement [115] or USDL [14], are employed to describe a service by means of their terms – such as guarantee terms or service description terms. However, these approaches do not provide mechanisms to specify the decision terms and their values in a succinct way. For this purpose, we have to describe the configurations one by one.

- Some authors have proposed variability modelling techniques [21, 88] to describe the configuration capabilities of HCSs in the cloud [69, 177]. These techniques have been widely used to represent the variability of software-intensive systems in the academia and the industry. Furthermore, they enable automated analysis techniques [19] to extract useful information from the models. However, the high configuration capabilities of HCSs exceed the modelling mechanisms provided by actual variability modelling.

[1] In this dissertation, we employ decision space and configuration capabilities as synonyms.

Regarding analysis, existing research proposals overlook the analysis techniques of HCSs [95]. As far as we know, there is no operations catalogue to automate the support of HCSs. The configuration of HCSs in the cloud has received some attention, although most of the times is not considered as a problem by its own [90, 97], and others the proposed approaches deal only with a small subset of the provider's decision space [55, 58, 109].

Finally, there is also room for improvement regarding the tool support. On the one side, the lack of automated support from the academia has been highlighted in the literature [86]. From the approaches which propose analysis techniques, only a few of them provide tool support and are still in their infancy, or are tied to particular cases or scenarios. On the other side, the support provided by the industry is insufficient. While the support offered by service providers is limited to the cost estimation of given configurations [3, 7, 130], commercial applications [38] often return false positives when searching for optimal configurations [73].

1.2 Thesis Goals

We have studied the state of the art with regard to all the issues described in the previous section, and we have identified a set of problems that constitute the goals to be achieved in this thesis, namely:

1. Analyse to what extent current proposals support the specification of HCSs configuration capabilities.
2. Provide a fully-fledged language to specify configurations of services (SYNOPSIS. SimplY a NOtation to sPecify Service configuratIonS).
3. Define a set of validity criteria for SYNOPSIS documents.
4. Analyse the analysis techniques currently used in HCSs-driven solutions.
5. Design and develop a set of analysis operations on SYNOPSIS documents.
6. Identify the usefulness of analysis operations for supporting real scenarios in the cloud.

These goals can be summarised in a main goal stated as follows:

Dissertation Goal

Improve the current support to develop HCS-driven solutions.

1.3 CONTRIBUTIONS

Our approach to address the aforementioned goals is highly inspired in the solutions proposed for the automated analysis in SLAs [101, 115, 138], and Business Process Management (BPM) [28, 47], and in particular for the description and automated analysis of variability models [17, 60, 154]. In these cases, the general approach was to extend existing models; to propose highly expressive notations, in order to represent such extensions; and to interpret analysis operations as queries on a formal representation of such models and their instances. The specific contributions of this thesis are depicted in Figure §1.1 and summarized in the following.

1.3.1 Contributions Regarding HCS Specification

Our first contribution (C1 in Figure §1.1) is the definition of a language to specify the decision space of a HCS (SYNOPSIS). As far as we know, we are pioneers at defining a Domain Specific Language to identify the decision space of HCSs. The main features of SYNOPSIS are:

1. Compliant with big HCS vendors: SYNOPSIS supports all the elements required to describe the decision space of HCSs provided by Amazon, Dropbox and Microsoft. Terms, dependencies, multi-items and and succinct mechanisms to describe pricing policies are the key elements in this point.

2. High and extensible expressiveness: SYNOPSIS allows specifying logical, relational and arithmetic relationships; and enumerated, boolean, integer and real terms by using a very expressive expression language. SYNOPSIS has been designed following the Semantics Priority Principle by [82], thus SYNOPSIS semantics assumes the existence of an ideal underlying variability analysis engine able to deal with any kind of variability element (feature, cardinality, attributes and

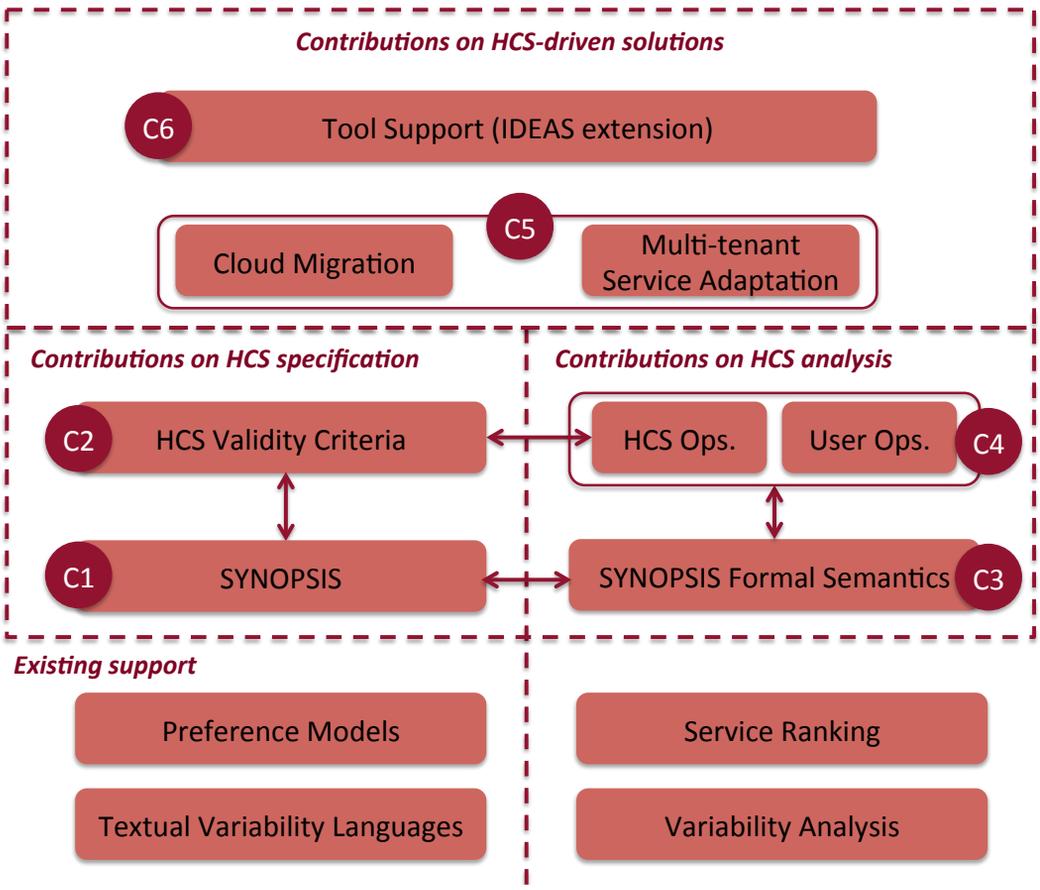


Figure 1.1: Contributions of this Dissertation.

multi-features). Current version of SYNOPSIS uses FAMA solver [158] – a tool for the automated analysis of feature models – and according to the criteria we have used to evaluate similar approaches from literature, we can claim that it is very expressive.

3. Domain-independent: the aforementioned extensibility and expressiveness endows SYNOPSIS with a very-high domain-independence.

4. Human-readable: SYNOPSIS provides a human-readable syntax.

The second contribution of this thesis (C2 in Figure §1.1) is the identification of a set of validity criteria for SYNOPSIS documents, which allows us to identify up to five kind of errors: redundant dependencies, dead values, false decision terms, false configurable service and inconsistent service.

1.3.2 Contributions Regarding Automated Analysis

The first contribution (C3 in Figure §1.1) aimed at providing support for automated analysis, consists of the definition of formal semantics for SYNOPSIS. This provides the SYNOPSIS constructions with precise meaning, and eases the automated extraction of information from documents written in SYNOPSIS (a.k.a. SYNOPSIS Documents or simply documents for short). In particular, SYNOPSIS semantics is based on *Stateful Feature Models (SFMs)* [154, 162].

Next, relying on SYNOPSIS semantics, we have defined and developed a catalogue of analysis operations, which are categorised in two groups (C4 in Figure §1.1): 1) operations for HCS providers to check and explain violations of SYNOPSIS validity criteria, and 2) operations for users to assist the decision making on HCSs. A reference implementation based on SFMs and their automated analysis tools [158] has been developed to support the execution of all the operations. Both groups of operations are ready to be used in HSC-driven solutions.

1.3.3 Contributions Regarding HCS-driven Solutions

The first contribution aimed at solving two well known problems in the HCS domain. On the one side, the migration of an on-premise infrastructure to the cloud, which has been interpreted as the search of the optimal cloud infrastructure configuration for given user requirements. And on the other side, the adaptation of a multi-tenant service to meet the changing preferences of the tenants, where the adaptation space of the service has been interpreted as the decision space of an HCS and the adaptation analysis is performed by means of user-side analysis operations.

The second contribution aimed at providing support for both human and software clients that are involved in the development of HSC-driven solutions (C6 in Figure §1.1). In this case, our solutions have been designed to be integrated in IDEAS (Integrated Development Environment for Service-driven Solutions (IDEAS). IDEAS provides a publicly-available user-friendly front-end which makes possible: 1) to edit SYNOPSIS documents assuring they comply the validity criteria, 2) to analyse some properties appealing for final and technical users. Such an environment is also available to be used by software clients both as a Java library, and a web service.

1.4 Thesis Context

This thesis has been developed in the context of the research group Applied Software Engineering (Ingeniería del Software Aplicada-ISA) of the University of Seville, and specifically on the context of Software Product Lines and Automated Analysis research area. As a holder of a pre-doctoral research scholarship from the SETI (*reSearching on intElligent Tools for the Internet of services*) national research project, the research has been developed in the scope of this project. Besides SETI project, there is a number of research projects that have made this dissertation possible:

- THEOS, *Tecnologías Habilitadoras para EcOsistemas Software*: In the context of this regional project, we propose the description and analysis of HCS as variability models
- COPAS, *eCosystems for Optimized Process As a Service*: In the context of this regional project, we propose the support of multi-stakeholder HCS configuration processes.
- TAPAS, *Tecnologías Avanzadas para Procesos como Servicios*: In the context of this national project, we propose the definition of *Configurable Service Models (CSMs)* to describe the configuration capabilities of HCSs.

Additionally, the PhD candidate has completed three research stays:

1. *A two-month research stay in Universidad Politécnica de Valencia*, under the supervision of professor Vicente Pelechano. June - July 2012.
2. *A three-month research stay in Cardiff University*, under the supervision of professor Omer F. Rana. September - December 2012.
3. *A three-month research stay in Lero*, the Irish Software Engineering Research Centre, invited by professor Mike Hinchey. September - December 2013.

1.5 Structure of this Dissertation

Part I. Introduction. This chapter provides an overview of the contributions of this dissertation.

Part II. Background. Chapter §2 summarises the background concepts for HCS descriptions. Chapter §3 details the background concepts for HCS and variability automated support.

Part III. Our contribution. Chapter §4 defines HCSs and our support for their specification. Chapter §5 describes our proposal for the automated analysis of HCSs.

Part IV. Validation. Chapter §6 presents our first validation scenario, in particular a migration to the cloud assisted by means of modelling and automated analysis techniques. Chapter §7 presents our second validation scenario, in particular a reconfiguration scenario in a multi-tenant service, assisted by means of automated analysis techniques.

Part V. Final Considerations. Chapter §8 briefly revises the contributions in this dissertation and explores the opportunities that arise from this work.

Part II

Background

HCS Modelling

Description is what makes the reader a sensory participant in the story.

Stephen King (1947),
Writer

In this chapter, we detail the foundations of Highly-configurable Services. Section §2.1 describes the concept of Highly-configurable Service and illustrates it with a couple of examples. In Section §2.2 we analyse the state of art for the representation of the decision space of services, while in Section §2.3 we do the same for the user needs. Section §2.4 ends the chapter.

2.1 HIGHLY-CONFIGURABLE SERVICES

The concept of HCS was firstly introduced by Lamparter et al. [94] for web services. They consider an HCS as a web service with multiple configurations C, where each configuration $C_j \subseteq C$ is mapped to a real number P_j that represents the price. This HCS concept has several shortcomings to be employed for a wider scope. First, the different configurations of the service are represented as attribute-value pairs, but the authors do not present the way to define the attribute domains. Second, there is no way to define dependencies among the different attributes, so the different service configurations must be defined in an extensive form. And finally, the proposed representation directly relies on the OWL formalism instead of a *Domain Specific Language (DSL)*, coupling to a particular resolution paradigm.

While Lamparter et al. [94] HCSs are related to web services, nowadays most of the existing HCSs are in the cloud. NIST [104] defines cloud computing as "*a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*" . Usually, cloud computing refers to both the applications and resources delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services [8].

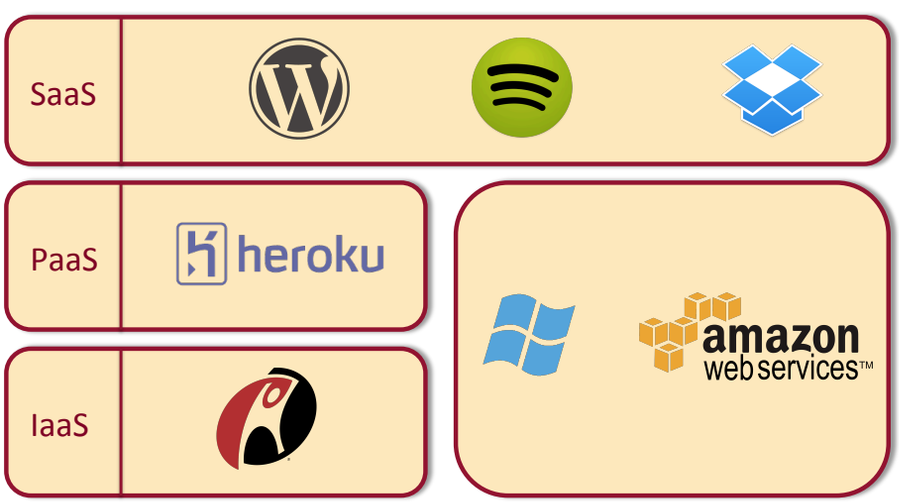


Figure 2.1: Cloud Service Models.

Cloud computing has given providers an enabling paradigm to massively deliver and customise different services. There are many cloud providers, such as Amazon,

IBM, Microsoft or Google, which offer a broad range of configurable services that are categorised in three service models (Figure §2.1): 1) *Infrastructure as a Service (IaaS)* provides processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software (e.g. Amazon EC2 [5] or Rackspace servers [131]); 2) *Platform as a Service (PaaS)* provides capabilities to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider (e.g. AppEngine [74] or Heroku [81]). 3) *Software as a Service (SaaS)* provides applications running on a cloud infrastructure, which are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface (e.g. Wordpress [178], Citrix's DaaS [33] or Dropbox [50]).

In the following, we present two examples of configurable services in the cloud: Dropbox as a configurable Storage as a Service, and Amazon EC2 as a Highly-configurable Computing as a Service.

2.1.1 Dropbox

Dropbox [50] is a file hosting SaaS that offers cloud storage, file synchronization, personal cloud, and client software. Dropbox allows users to create a special folder on their computers, which Dropbox then synchronizes so that it appears to be the same folder (with the same contents) regardless of which computer is used to view it. Files placed in this folder are also accessible via the Dropbox website and mobile apps.

Dropbox provides fewer configuration capabilities than other services. Dropbox users can choose among three different plans (basic, professional and business). Additionally, there are official and unofficial add-ons, mostly created by the Dropbox community. These add-ons are both in the form of web services such as SendToDropbox (which allows users to email files to their Dropboxes), or Mover (which facilitates online backup of FTP, Git, MySQL, and other services to Dropbox accounts).

However, the configuration capabilities of Dropbox have a strong impact on the delivered service. This is especially so in the case of the plans. While the basic plan provides reduced storage, backup and sharing options at no cost, professional and business plans increasingly improve these capabilities together with the service price.

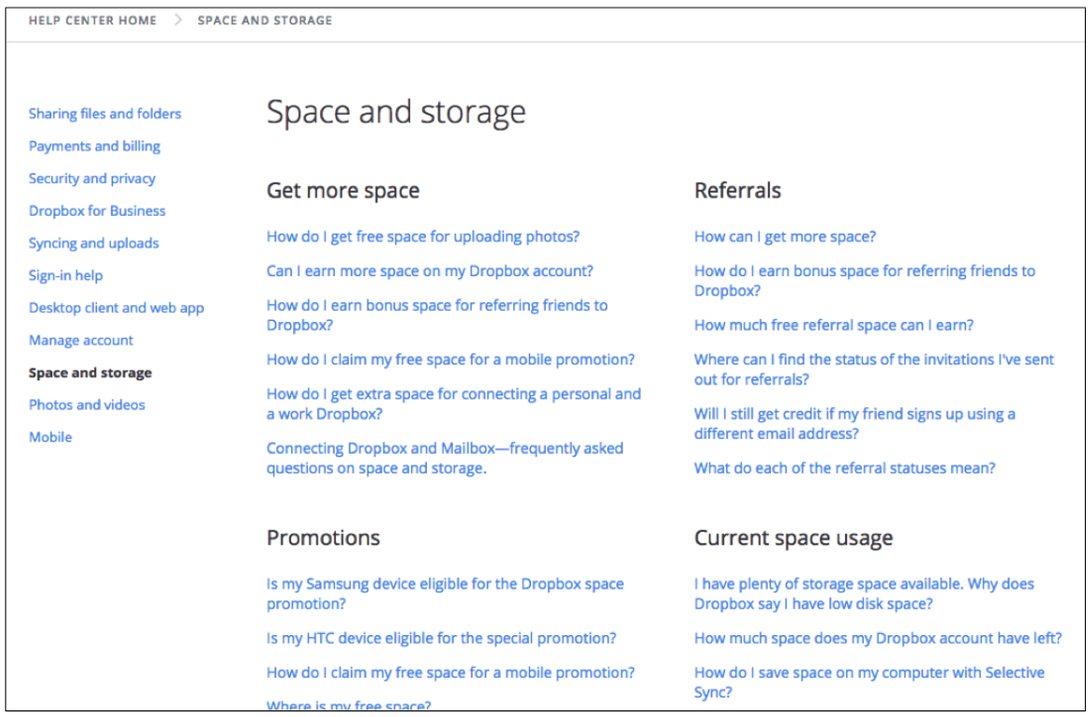


Figure 2.2: Excerpt of Dropbox description.

2.1.2 Amazon EC2

Amazon is one of the main cloud commercial providers. In particular, the Amazon Web Services division [4] provides services for computation [5], storage [6], databases or content delivery among others. Due to the wide range of services, several PaaS and SaaS providers like Heroku or Netflix run on top of the Amazon services. The service offering of Amazon is continuously evolving, and for this reason we work in this dissertation with a specific snapshot of the provided services from 12th June 2014.

We focus in particular on Amazon EC2, the computing cloud service of Amazon, to illustrate the configuration capabilities of an IaaS. Amazon EC2 provides "provides resizeable compute capacity in the cloud" [5] on pay-per-use basis, with different options to configure the aspects of the computing instances. Additionally, we can add extra storage to such instances with the EBS service. On the aforementioned date, Amazon provides six main configuration options for EC2:

- *Instance type*. EC2 offers 32 different instance types, grouped under different categories, depending on the purpose: General Purpose, Compute Optimized, GPU Instances, Memory Optimized and Storage Optimized, distinguishing also between current and previous generations. Each instance type has a specific RAM

size, a number of cores and disk storage [1].

- *Operating system*. Three different Linux distributions (Amazon Linux, Suse and Red Hat) and a Windows version (Windows Server) with an optional SQL Server in different flavours (Express, Web and Standard) are available.
- *Storage*. Amazon provides a fixed disk size for EC2 instances, depending on each instance type. If the default storage needs to be extended, the EBS service provides additional storage.
- *Geographic location*. Amazon offers 8 geographic locations, distributed among different areas of America, Europe, Asia and Oceania.
- *Purchasing mode*. EC2 instances may be purchased on demand or in a reservation way. While users pay per use in both modes, cost hour is lower for reserved instances in exchange for an upfront payment. In total, there are seven different purchasing modes.
- *Dedication*. Additionally, an EC2 instance may run on a dedicated machine, guaranteeing an additional isolation in exchange of an additional cost.

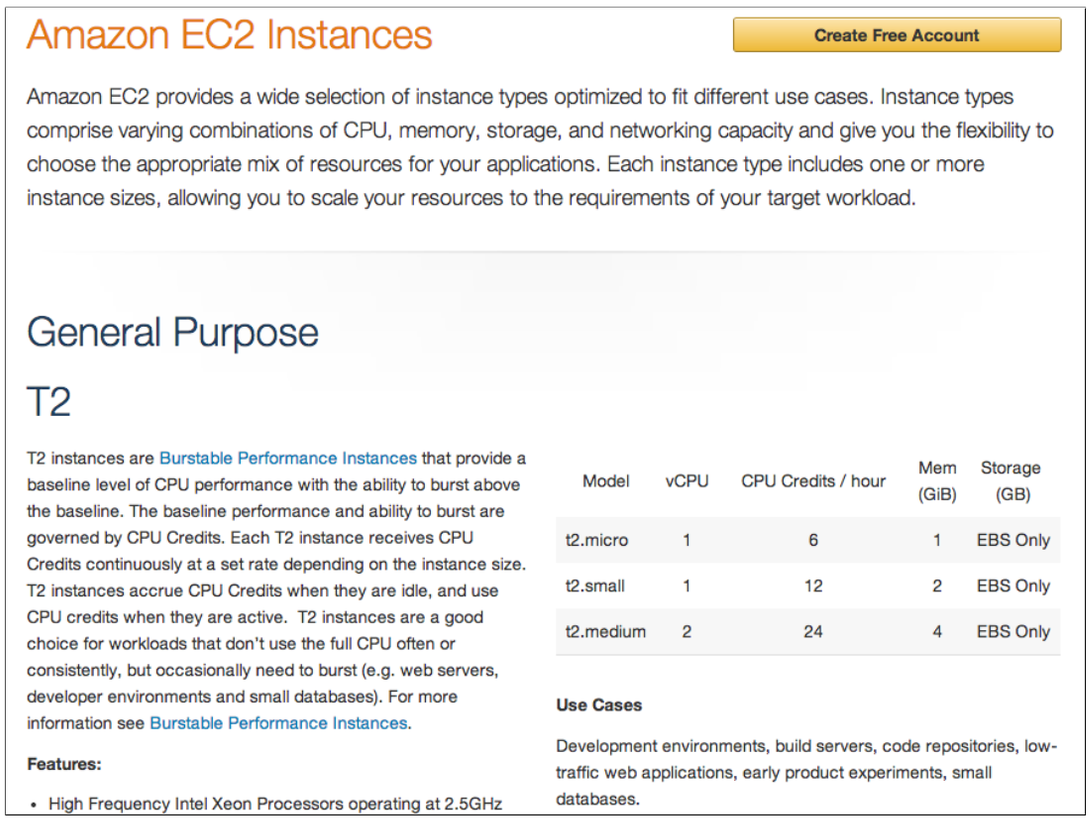
All these configuration options and values lead to 68544 potential different configurations. However, there are constraints that bound the number of available configurations. For instance, some instance types are not available for given geographic locations. In the same way, not all the instances can be reserved or dedicated. Considering all these constraints EC2 presents 16991 available configurations – this number will be explained during the next chapters.

2.2 Decision Space Representation Techniques

2.2.1 Structured Natural Language

Natural language is the most common way of representation by commercial providers such as Dropbox [50], Amazon [5] or Microsoft [112]. Figure §2.2 and Figure §2.3 show examples of the description of some instances of Amazon EC2 and Dropbox

[1] We ignore micro instances, since they are intended for short CPU burst purposes and their performance is highly variable and unpredictable.



Amazon EC2 Instances

Create Free Account

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instance types comprise varying combinations of CPU, memory, storage, and networking capacity and give you the flexibility to choose the appropriate mix of resources for your applications. Each instance type includes one or more instance sizes, allowing you to scale your resources to the requirements of your target workload.

General Purpose

T2

T2 instances are Burstable Performance Instances that provide a baseline level of CPU performance with the ability to burst above the baseline. The baseline performance and ability to burst are governed by CPU Credits. Each T2 instance receives CPU Credits continuously at a set rate depending on the instance size. T2 instances accrue CPU Credits when they are idle, and use CPU credits when they are active. T2 instances are a good choice for workloads that don't use the full CPU often or consistently, but occasionally need to burst (e.g. web servers, developer environments and small databases). For more information see Burstable Performance Instances.

Features:

- High Frequency Intel Xeon Processors operating at 2.5GHz

Model	vCPU	CPU Credits / hour	Mem (GiB)	Storage (GB)
t2.micro	1	6	1	EBS Only
t2.small	1	12	2	EBS Only
t2.medium	2	24	4	EBS Only

Use Cases

Development environments, build servers, code repositories, low-traffic web applications, early product experiments, small databases.

Figure 2.3: Excerpt of Amazon EC2 description.

capabilities, respectively. The lower the level of the service in the service model, the more detailed the configuration capabilities are in the description. On the opposite side, the higher the level of the service, the more oriented to marketing the description is.

Dropbox [50] is described by means of natural language. Although this service has only one main configuration option with three values – basic, professional and business plans –, the decision space is not really well described. In order to understand the exact impact of each plan in the rest of configurable terms – storage, sharing options or backup – it is necessary to carefully read and navigate through the whole documentation (an excerpt is shown in Figure §2.2).

Amazon EC2 is described in its website [5] by means of natural language and tables. They use text for describing the big picture and marketing aspects, but also some constraints; and tables to show the specific configuration options, a.k.a. decision terms, values and details. In this sense, the different decision terms and values are scattered among several pages – for instance Amazon groups instance types in two categories named new generation and previous generation, each in a different page.

Most of the decision space of EC2 is exhaustively described, but the use of natural language leads to some ambiguities and unclear aspects. While some non available configurations are explicit, by means of a N/A mark in the tables – for example, a cc2.8xlarge light reserved instance in Sao Paulo – , others non available configurations simply are excluded from the tables – for example, a i2.8xlarge light reserved instance for the same location. Besides, there are other constraints that are described using natural language. For example, if we contract a heavy reserved instance we are forced to pay for a 24x7 usage for he contracted period.

As far as we know, neither providers nor academia propose specific DSLs to describe the decision space of HCSs. While, as we have seen, providers employ natural language, in the literature the work of Lamparter et al. [94] is the closest approach to a DSL. Although natural language is flexible, highly expressive and comprehensible by most of the consumers, it also present serious drawbacks. Its intrinsic ambiguity, a lack of structure and exhaustiveness and the heterogeneous descriptions made by service providers harden the fully understanding of the service and a posterior automated processing.

2.2.2 Variability Modelling

Variability is a property of software systems that provides them with the ability of being customized according to the specific user needs. Operating systems like Debian OS or Android, or *Software Product Lines (SPLs)* are examples of the so-named *variability-intensive systems*. While some authors prefer to use the term "*highly-configurable systems*", others prefer the aforementioned variability-intensive systems. *Variability Models (VMs)* are used to describe the common and variable aspects of variability-intensive systems. They represent all the possible configurations of the system, often in terms of functional features [88], although they can also describe extra functional properties [18].

In the same way that VMs are used to represent the configurations of variability-intensive systems, they can be employed also to describe the configuration capabilities of HCSs. In general terms, a VM is composed by a set of features – or elements – which are interrelated among them by means of relationships. While VMs represent the functional features of SPLs and their relationships, they can describe the decision terms of HCSs and their dependencies.

Feature Models

One of the most used VMs are the so-called *Feature Models (FMs)*. FMs represent the commonalities and variabilities of a system in terms of features, which are distinctive characteristics a user can observe [88]. A FM can de defined as follows:

Definition 2.1 - Feature Model.
A FM is a tuple (F, P) such that F is the set of features of the system and P the set of available configurations defined as a subset of the powerset of features, i.e. $P \subseteq \mathcal{P}(F)$, such that a configuration is defined by a set of features in F.

Let us take a *Smart Home Systems (SHSs)* as an example. Its set of features F and a possible configuration C_1 (see Figure §2.4) can be defined as follows:

$$
\begin{aligned}
F = \; & \{SmartHome, Lighting, ControlSystem, CellPhone, ControlPanel, \\
& AntitheftAlarm, Internet, Ethernet, 3G, WiFi-b/g, WiFi-n, MoviePlayers, \\
& HDTV42, HDTV32, PCPlayer, Contents, VideoOnDemand, Providers, \\
& Cache, DMS\} \\
C_1 = \; & \{SmartHome, Lighting, CellPhone, Alarm, HDTV42, \\
& HDTV32, DMS, VideoOnDemand\}
\end{aligned}
$$

In order to define the set of configurations, a FM comprises a set *relationships* that limits the allowed feature combinations, so that a configuration must satisfy all the relationships. Relationships in a FM are mainly hierarchical. Any FM has a *root* feature that represents the whole functionality of any configuration. The root feature is refined in child features, which decompose the behaviour or functionality of the root feature into subfeatures, which describe the scope of the root feature in more detail. This refinement process is repeated for the child features to conform a tree-like structure. Although the hierarchical structure helps to represent the feature refinement, it can hinder the representation of restrictions that affect features in different branches of the tree. In these circumstances, *cross-tree constraints* can be used. *Feature diagrams* [133, 142] are probably the most used graphical representation of FMs. Figure §2.4 presents a feature diagram for a SHSs.

FMs have evolved in time adding new elements to the set of features and relationships. So Czarnecki et al. [42] and Riebisch et al. [133] propose *Cardinality-Based Feature Models (CBFMs)* as an evolution of basic FMs that introduce cardinalities. They allow

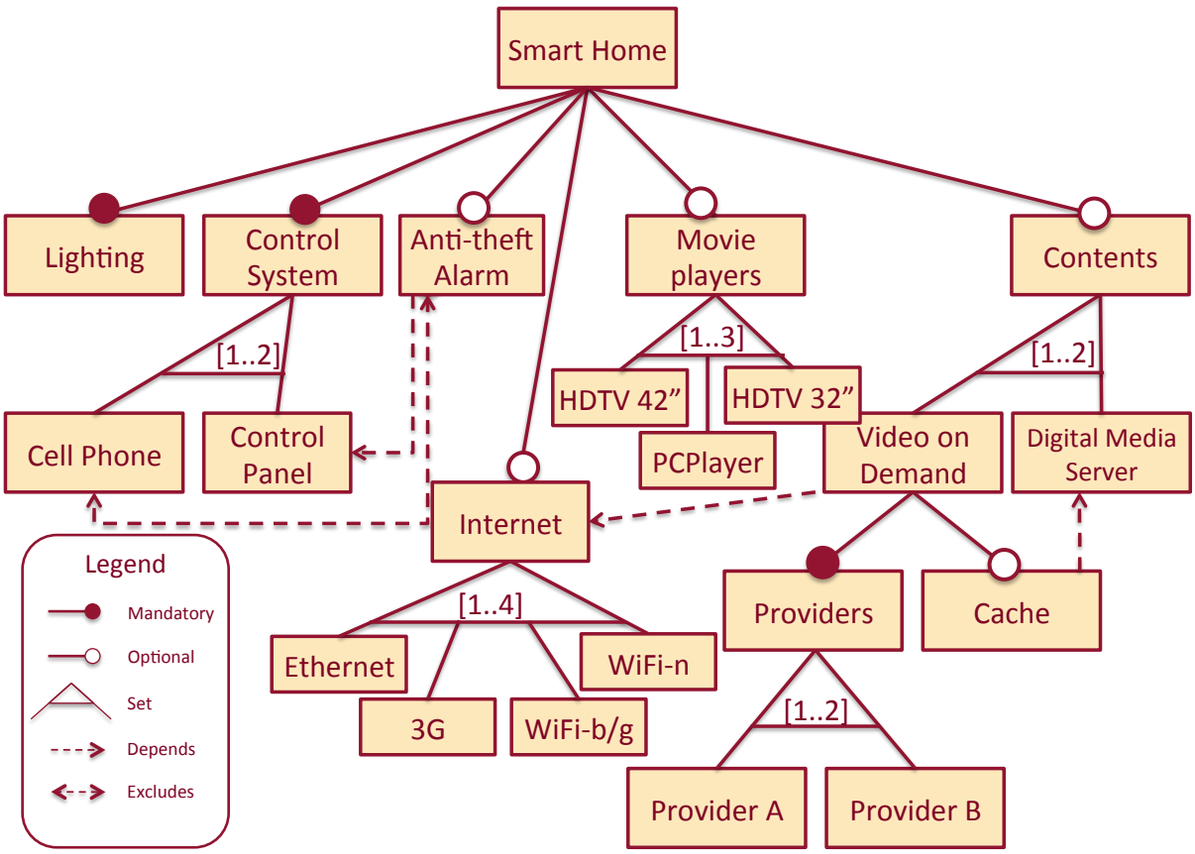


Figure 2.4: Feature diagram representing a smart home.

to group a set of features and assign them a cardinality that denotes the number of features (cardinal) that can be selected at the same time. CBFMs increase the succinctness of the model, and they are as expressive as basic FMs, as Schobbens et al. [142] proved.

CBFMs use the following kinds of hierarchical relationships in FMs:

- **Mandatory**: a mandatory relationship affects a parent and child feature. It forces the child feature to appear in a configuration whenever its parent feature does. For example, any SHS must have `lighting` and `controlSystem` features. If a `videoOnDemand` feature is selected, then `providers` must also be selected.
- **Optional**: a child feature connected to a parent feature by means of an optional relationship may be optionally selected whenever its parent feature is. For example, the `antitheftAlarm` and `Internet` connection are optional features in a SHS.
- **Set relationship**: set relationships affect a parent feature and a set of two or more child features. It contains a set of natural numbers or *cardinality* that constraints

the number of child features to be selected in a configuration whenever its parent feature is selected. For example, if the *Internet* feature is selected, then 1 to 4 features must be selected from `Ethernet`, `3G`, `WiFi-b/g` and `WiFi-n` features.

Alternative relationships can be interpreted as a particular case of set-relationship with a $[1..1]$ cardinality, where only one child feature may be selected in a configuration at the same time if the parent feature is selected. In turn, *or-relationships* are those set relationships whose cardinality is $[1..N]$ such that N is the number of child features. For example, the `InternetConnection` feature is the parent in an or-relationship.

Besides hierarchical relationships, *cross-tree constraints* break the tree-like structure to represent non-hierarchical relationships. The most used cross-tree constraints are:

- **Dependency**: a feature depends on another feature if the second one must be part of a configuration whenever first one is selected. For example, the `cache` feature requires for a `digitalMediaServer` feature to store video and the `antitheftAlarm` requires a `controlPanel` for de/activation.
- **Exclusion**: two features exclude themselves if both of them cannot be part of a configuration at the same time. For example the `antitheftAlarm` feature is incompatible with a `cellPhone` feature for security reasons.

Besides features, FMs can collect additional information by using the so-called *attributes*. An attribute represents relevant information such as feature development cost, versions, RAM consumption, performance or technological requirements. FMs that use attributes are known as *Extended Feature Models (EFMs)* [18]. Figure §2.5 shows an example of a FM with attributes. It adds information regarding Internet bandwidth to an excerpt of the FM in Figure §2.4. Each kind of connection provides a different bandwidth. Since more than one connection can be chosen, the maximum available Internet bandwidth in the SHS is the maximum bandwidth provided by each chosen connection.

An EFM may contain constraints that affect attributes which reduce even more the set of configurations an EFM describes. So for example, if a constraint sets the `Internet` max bandwidth to at least 80 Mbps then it forces any SHS to have at least an `Ethernet` or `Wifi-n` connection since they are the only features providing such bandwidth. Along this dissertation, we will refer using the term FM to FMs, CBFMs and

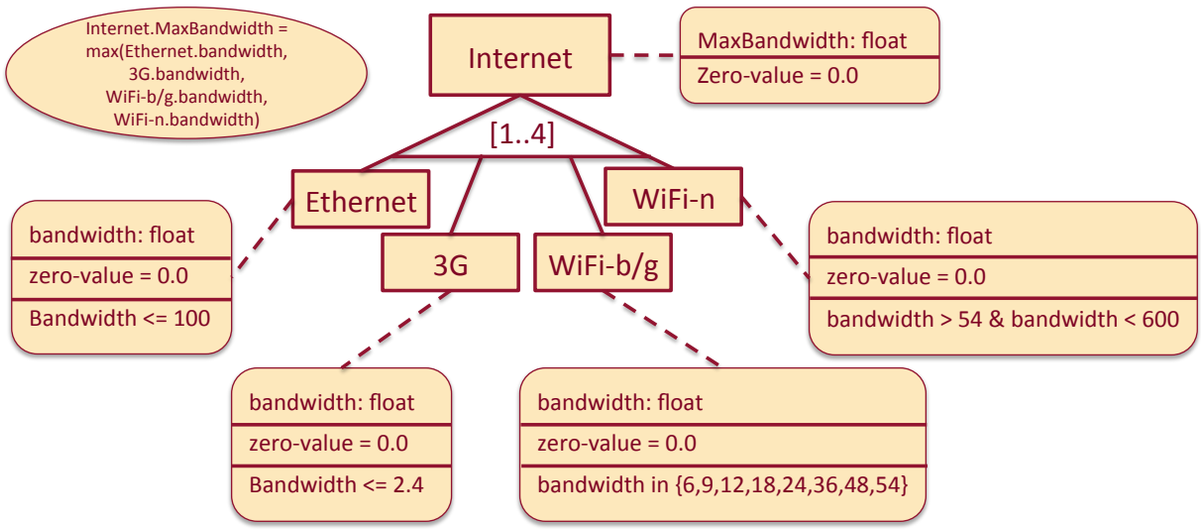


Figure 2.5: Excerpt of a feature diagram representing an extended FM

EFMs indistinctly. Since the latter is an extension of the former, we consider both the same kind of model for the sake of simplicity, as in the same way many authors do [9, 141, 146].

Multi-features

The concept of multi-feature – a.k.a. cloneable feature – is common in variability-intensive systems. Features in a FM are unique, which means that no other feature can refer to the the same functionality. However, with multi-features, it is possible to create more than one feature instance, although the concept of instance still has an ambiguous interpretation. The FM in Figure §2.6 shows an example where the ambiguity arises. A car must have 4 tires, each of which can be hard or soft. Must all the tires be either soft or hard? Or is it possible to combine them anyhow?

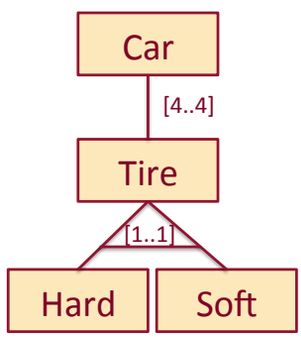


Figure 2.6: Feature cardinalities may lead to ambiguous situations

Several authors have made proposals to represent multi-features. Riebisch et al.

[133] proposed a feature cardinality relationship, where a parent and a child feature are linked by a cardinality which indicates the number of valid instances of the child features that can be selected in a configuration. The cardinality [0..1] is equivalent to an optional relationship and [1..1] to a mandatory relationship. Later, Czarnecki et al. [43] proposed cardinalities both for group relationships and also for single features. However, the multi-level configuration process that they present in this work lack support for the latter cardinalities.

Michel et al. [110] formalised the concept of clone in feature diagrams, and differentiated between group cardinalities and feature cardinalities. While the former describes the number of children that can be selected in a group relationship, the latter refers to how many times a given feature can be repeated. In view of the clones, they redefine the concept of product, since a feature can be included more than once in a product.

Cordy et al. [39] introduced the current term – multi-feature – and approach this using their TVL language. The authors identify the ambiguities introduced by cardinalities, that can be defined on group relationships or single features, and propose an array-based syntax to refer to multi-features. They also introduce the notion of context, i.e. a subtree of the model that may include multi-features, and the existential and universal quantification operators for the definition of constraints.

Textual Variability Languages

Besides FMs as general representations, there are specific languages to realise the variability modelling capabilities of these models. We describe the main textual variability languages identified and compared in the work of Eichelberger and Schmid [53]. Most of the following descriptions are extracted from that paper:

- The Feature Description Language (FDL) [48] was constructed by applying design principles of Domain-Specific Languages (DSLs). In addition to the syntax, the authors specify a feature diagram algebra.

- The tree-grammar approach [15] represents cardinality-based feature diagrams using (iterative) tree grammars. In particular, Batory represents constraints in terms of propositional formulae.

- The Variability Specification Language (VSL) [132] aims for the integration of prominent feature modeling approaches with configuration links and variable

entities. In VSL, configuration links are a specific form of references. Variable entities enable the modelling of predefined chains and networks of configurations interrelated by configuration links.

- The Simple XML Feature Model (SXFM) is a XML based language which enables its automated analysis. SXFM is used in the feature model repository Software Product Lines Online Tools (S.P.L.O.T) [105].

- FAMILIAR [1] aims for defining, combining, analysing and manipulating feature models. Due to the scope of the dissertation, we focus on the variability modelling concepts rather than on the scripting language capabilities.

- The Textual Variability Language (TVL) [35] is intended to overcome the shortcomings of graphical notations like feature diagrams, which are usually inferior to text formats in terms of tool support and syntax richness. TVL allows to define attributes and complex constraints, and provides different syntactic sugars to ease the definition of FMs. TVL*, an extended version of TVL, provides support for real attributes and multi-features [39].

- μTVL (micro TVL) [34] is a subset of TVL which aims at core feature modeling. Basically, μTVL drops some types and changes some semantics. Further, μTVL enables multiple feature trees in one model.

- The CLAss, FEature, Reference approach (Clafer) [12] combines meta-modeling (of classes) with first-class support for feature modelling. Clafer aims at a minimal number of concepts with a uniform semantics. Clafer provides specialization and extension layers via constraints and inheritance, explicit containment, cardinalities, multiple instances and (object) references.

- VELVET [137] aims for support for separation of multi-dimensional concerns in feature-based variability modeling. Some basic concepts in VELVET are inspired by TVL. VELVET provides several mechanisms to combine the individual models, i.e., concerns, into a common variability model.

- The INDENICA Variability Modeling Language (IVML) [127] is developed in the EU-funded project INDENICA on customization and integration of service platforms. IVML is designed as a scalable, textual variability modeling language, which supports the variability modelling requirements that are relevant to support the customization of complex service platform ecosystems.

Variability Modelling Approaches			
	Expressiveness		
	Elements	Constraints	Language Support
Basic FMs [88]	f	b	-
CB FMs [42]	f,c,m	b	-
FDL [48]	f,(c),m	b	✓
Batory [15]	f,m	l	∼
VSL [132]	f,c,a,m	b,(l),t	✓
SXFM [105]	f,c	(l)	✓
FAMILIAR [1]	f	l	✓
TVL [35, 39]	f,c,a,m	l,r,t,(m)	✓
μTVL [34]	f,c,a,m	b,l,r,t	✓
Clafer [12]	f,c,m	l,r	✓
VELVET [137]	f,(c),a	l,r	✓
IVML [127]	f,c,a,m	l,r,t	✓
FAMA [84, 158]	f,c,a	b,l,r,t	✓
SFMs [154, 162]	f,c,a	b,l,(r),(t)	-

f=features, c=cardinalities, a=attributes, m=multi-features, b=basic cross-tree constraints, l= logical constraints, r=relational constraints, t=arithmetic constraints, (x)= implicitly supported, ✓=support, ∼ =partial support, - =no support

Table 2.1: Comparison of main variability modelling approaches.

- FaMa Framework [158] provides a text language for the definition of cardinality-based FMs. FaMa is a tool for the automated analysis of FMs, so the input models for the analysis are defined in this format.

In Table §2.1 we compare the presented variability modelling approaches . This comparison is based on the work of Eichelberger and Schmid [53], but we focus in particular on the expressiveness. The expressiveness is measured in terms of elements and constraints. Four different types of elements – features (f), cardinalities (c), attribute (a)s and multi-features (m) – and constraints – basic cross-tree constraints (b), logical constraints (l), relational constraints (r), arithmetic constraints (t) and constraints on multi-features (m) – are identified. While most of the times these elements are supported explicitly, sometimes their support is implicit. If such support is implicit we employ parentheses surrounding the element, and in the case it is explicit we avoid the parentheses.

As Table §2.1 shows, there is no comprehensive approach which successfully deals with all the presented aspects. In terms of modelling and expressiveness, TVL [34] can be considered the most complete approach. However, it presents lacks for describing

relationships among different instances. We have tried to model with TVL a simple computing HCS with a discount calculated on the total cost of all the hired instances. The resulting model (Listing §2.7) shows that there is no way to aggregate the total cost. SFMs [154, 162] are specialised in the variability configuration and provide support for automated analysis, although they do not provide specific textual languages to describe the variability and make decisions on it.

Figure 2.7: An IaaS described using TVL.

```
root ComputingHCS{                                                                    1
        ComputingInstance [*..*]{                                                     2
                OperatingSystem group [1..1]{                                         3
                        Windows, Linux                                                4
                },                                                                    5
                InstanceType group [1..1]{                                            6
                        Small, Medium, Large                                          7
                },                                                                    8
                Region group [1..1]{                                                  9
                        US, Europe                                                    10
                }                                                                     11
                // Instance attributes                                                12
                int cost, hours, costHour;                                            13
        }                                                                             14
        // HCS attributes                                                             15
        int discount;                                                                 16
        int totalCost;                                                                17
        // Requires constraint                                                        18
        Windows -> US;                                                                19
        // Discount constraint                                                        20
        (forAll(ComputingInstance){Region.US}  || forAll(ComputingInstance){Region.Europe})}  21
        && (ComputingInstance[1].cost + ComputingInstance[2].cost + ... > 10 000)     22
        ->                                                                            23
        ComputingHCS.discount == totalCost*0.05;                                      24
}                                                                                     25
```

Variability Modelling of HCSs

VMs are trending for the representation of cloud HCS during the last years [69, 128, 176]. HCS descriptions based on variability modelling present several benefits. First, VMs provide a succinct way to represent – and to depict graphically – the decision terms of services. Second, these descriptions are unambiguous, since VMs often present a clear notation – as shown with variability languages in Section §2.2 – and have been formalised. And third, VMs enable automated analysis techniques that can be employed to automate the validation and configuration, as in the case of FMs [18], which provide a catalogue of analysis operations in the *Automated Analysis of Feature Models (AAFM)* [19].

Several of the most relevant works about VMs and cloud HCSs employ FMs for the modelling. Quinton et al. [128] propose FMs for the configuration process of cloud environments, considering features as deployable artefacts and automating the deployment of the configurations. Schroeter et al. [144] also propose the use of FMs for the configuration of cloud services, together with the definition of a configuration process model based on staged configurations.

The approach of Wittern et al. is particularly related to this dissertation. Wittern et al. [177] identify the need to assist the configuration of cloud services, proposing the so-named Cloud Feature Models (CFMs, based on EFMs) and a Cloud Service Selection Process. CFMs, which incorporate the aggregation of attributes, are used for representing abstract relevant decisions and concrete cloud offers, and for describing the specific requirements of decision-makers. For the selection, they propose a process, using Business Process Model Notation notation, composed of several tasks, going from the service modelling to the ranking of different configurations. In a later approach, Wittern and Zirpins [176] present a similar proposal, the Service Feature Modelling. However, this work is more oriented to general services and focuses on the preferences-based ranking of the alternatives.

2.3 User Needs Representation Techniques

2.3.1 Configuration Models in Variability

The process by which one or more users define the configuration that best fits their needs by making successive decisions on a particular FM is called a *configuration process*. The decisions made by users in a configuration process are collected in a *configuration*. Users usually express their decisions in terms of feature selections or removals, i.e. which features must be part of a configuration or left aside. In EFMs, users can also make decisions on attributes, restricting the domain by means of arithmetic and logical constraints. Four main types of configurations are considered in the literature, as shown in Figure §2.8:

- **Individual**: in this case – firstly considered by Kang [88] – only one user participates in the configuration process. A new decision cannot contradict any decision that has been previously made by the same user. Thus, a feature that is already selected cannot be removed later.

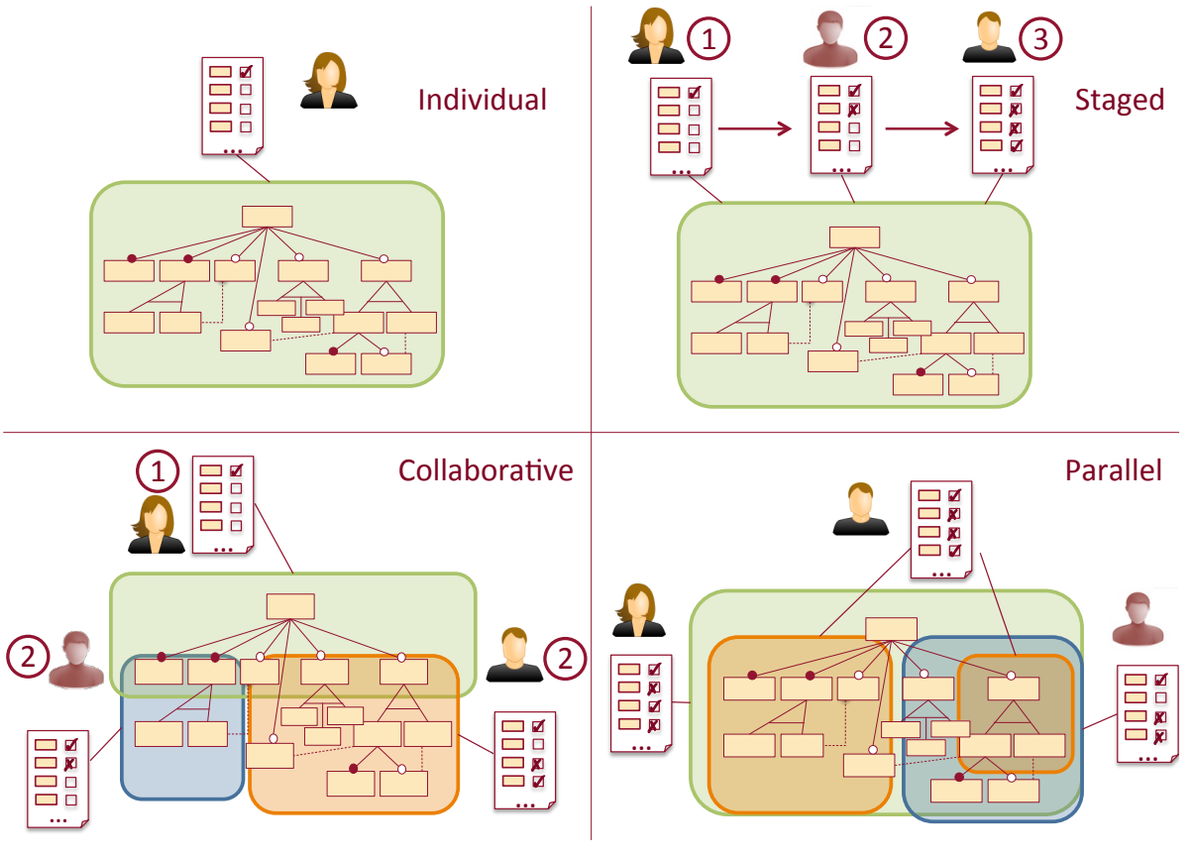


Figure 2.8: The main four types of variability configurations processes.

- **Staged**: in this case, proposed by Czarnecki et al. [42], two or more users perform the configuration process in stages. At each stage one user makes decisions, updating the configuration of the model. It is possible that a user wants to change a decision on a feature that has already been selected or removed by another user in a previous stage. In this sense, the contradiction must be annotated so that the user that selected or removed the features previously is informed so that the conflict can be solved among users.

- **Parallel**: in this case, considered by White et al. [174], two or more users make decisions at the same time and in any order, enabling users to contradict each other. So for example, a user might remove a feature that is being selected by another user at the same time.

- **Collaborative configuration**: this process, proposed by Mendonça et al. [106], lies halfway between staged and parallel configuration processes. Multiple users are allowed to make decisions in parallel but only on a specific part of the FM. These parts are previously calculated so that conflicts among decisions are avoided.

Both staged [41] and and collaborative [107] configuration enable mutual exclusion mechanisms to avoid conflicts. While staged configurations avoid potential conflicts by a sequential process, where a users only can make decisions on the undecided elements of the system; collaborative configuration propose a refinement on the staged configuration: multiple users can configure at the same time if they do it in disjoint areas with no dependencies among them.

However, the parallel configuration of FMs is a non tackled challenge. The usual way to deal with multi-user configuration scenarios is by means of mutual exclusion mechanisms, in order to avoid conflicts. However, for scenarios where there are shared resources in real time, a mutual exclusion approach is not possible. This kind of scenarios are usual for smart or pervasive systems, or even in multi-tenant services.

Configuration Models

Users usually express their decisions in terms of feature selections or removals, i.e. which features must be part of a product or left aside. In order to collect these decisions *Configuration Models (CMs)* are structured as follows:

Definition 2.2 - Configuration model.
Given a FM as a tuple (F, P), a configuration for this FM, denoted as γ_{FM} is a three-tuple of the form (S, R, U) in which S, R and U denote three disjoint finite sets of selected, removed and undecided features respectively in such a way that all the features of F must belong to one and only one of these three sets, i.e.

$$\gamma_{FM} = (S, R, U) \Leftrightarrow F = S \cup R \cup U \text{ and } S \cap R \cap U = \emptyset$$

When a configuration process starts, all the features are in the undecided set to indicate that no decision has been made about them. Whenever a user selects a feature, it is moved from the undecided set to the selected set. If a feature is discarded then it is moved from the undecided set to the removed set. Depending on the distribution of features among the three sets, we can define a configuration state as follows:

Definition 2.3 - Configuration states.
A configuration γ_{FM} is *partial* if there are still decisions to be made, otherwise it is said to be a *full* configuration. Both states are denoted as $partial(\gamma_{FM})$ and $full(\gamma_{FM})$.

$$\begin{aligned} partial(\gamma_{FM}) &\Leftrightarrow U \neq \emptyset \\ full(\gamma_{FM}) &\Leftrightarrow U = \emptyset \end{aligned}$$

Whether partial or full, a configuration is said to be *valid* if there is at least one product in the corresponding FM that contains all the features in the selected set and no feature in the removed set; otherwise it is said to be an *invalid* configuration. Both states are denoted as $valid(\gamma_{FM})$ and $invalid(\gamma_{FM})$.

$$valid(\gamma_{FM}) \Leftrightarrow \exists p \in P \cdot S \subseteq p \wedge R \cap p = \emptyset$$
$$invalid(\gamma_{FM}) \Leftrightarrow \neg \exists p \in P \cdot S \subseteq p \wedge R \cap p = \emptyset$$

For example, the following configuration for a SHS is partial since some features have already been selected, some others have been removed and others are still to be decided what to do with them:

$$S = \{SmartHome, Lighting, ControlSystem, CellPhone, Internet, Ethernet\}$$
$$R = \{ControlPanel, MoviePlayers\}$$
$$U = \{Antitheft, Contents, VideoOnDemand, DMS, ...\}$$

Full configurations are the result of a completed configuration process. For the SHS example, the following configuration is full:

$$S = \{SmartHome, Ligthing, ControlSystem, CellPhone, Internet, Ethernet\}$$
$$R = \{ControlPanel, MoviePlayers, Antitheft, Contents, Videoondemand, ...\}$$
$$U = \emptyset$$

The following example describes a valid partial configuration for the SHS SPL since there exists at least one product with the selected features and without the removed features:

$$S = \{SmartHome, Lighting, ControlSystem, Cellphone, Internet, Ethernet\}$$
$$R = \{ControlPanel, Antitheftalarm, Contents, ...\}$$
$$U = \emptyset$$

The following example describes an invalid configuration since *anti-theft alarm* depends on a *control panel* feature, which is removed:

$$S = \{SmartHome, Lighting, ControlSystem, Cellphone, AntitheftAlarm\}$$
$$R = \{ControlPanel\}$$
$$U = \{Contents, Internet, ...\}$$

Both, valid and invalid configurations can be defined in terms of relationships satisfiability. A relationship is said to be satisfied by a configuration if the selected and

removed features correspond to the relationship expected behaviour. In those terms, a valid configuration is the one that satisfies all the constraints, while an invalid configuration violates at least one configuration in the FM.

It is frequent in staged and parallel configuration processes to allow users to contradict previous decisions. In this case, features can be selected and removed at the same time, i.e. the set of features in conflict (C) is $C = S \cap R$. These conflicts must be solved in order to accomplish the configuration process, existing several proposals to repair them [122, 174].

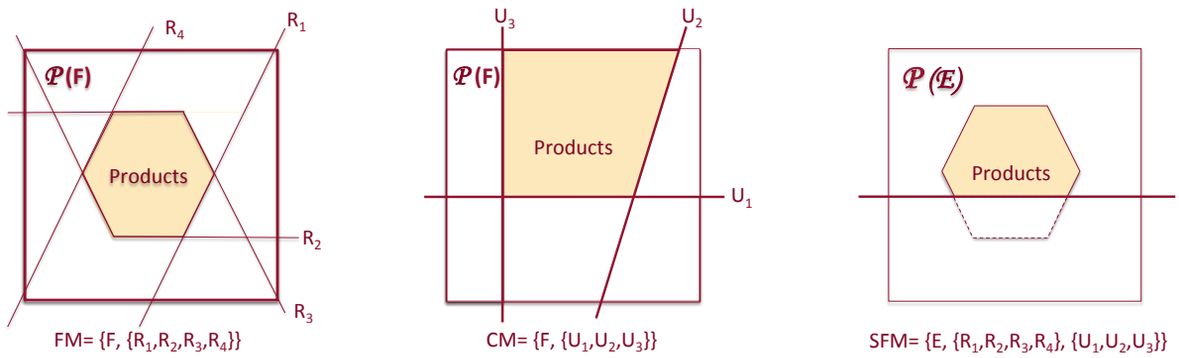
There are proposals of configuration models in the literature. Some of them are integrated with variability modelling mechanisms [1, 41, 132, 137] previously described in Section §2.2, and others are related to configuration processes [41, 107]. Additionally, some other proposals not originally intended only for configuration can be considered, such as White et al. [173] which present a technique for selecting highly optimal feature sets, or Roos-Frantz et al. [135] which present a similar approach for the optimisation of attributes, but in this case for orthogonal variability models.

Of particular interest are the approach from Zhang et al. [180] and Asadi et al. [9]. Zhang et al. [180] propose to extend the decision making in variability configuration to quality attributes. And Asadi et al. [9] detail a novel configuration approach, based on preferences about the user requirements. Users can define their requirements on functional features and attributes, and can also define an order for their constraints, which will establish the relative importance if it is not feasible to satisfy all the requirements.

2.3.2 Stateful Feature Models

Trinidad [154, 162] proposes incorporating FMs and CMs together in a single model, the SFM. The resulting model stores together the set of elements E – features, attributes and cardinalities –, and two sets of constraints, one for FM relationships and another one for user decisions (see Figure §2.9). SFMs can be considered as fully-configurable FMs, able to represent user decisions on features, attributes and cardinalities, and distinguishing also between user and automatic decisions.

SFMs rely on a new vision of configurations as an assignment of states to elements. In FMs, each configuration is described as a different subset of features. In SFMs, all the configurations share the same set of elements, denoted by a non-empty set $E = \{E_1, ..., E_n\}$ where each E_i is an element in that model, either features or cardinalities. A configuration is defined as an assignment of states to every element in E such that



(a) Feature model (b) Configuration model (c) Stateful feature model

Figure 2.9: A visual metaphore of FMs, CMs and SFMs

each element has a set of available states that depends on the kind of element. This way, features have selected or removed states to indicate their presence or absence in a configuration; attributes have allowed or removed ranges; and a cardinality has a cardinal value as state to indicate the number of features that are in a selected state within a set relationship.

The SFMs takes extensibility as a major concern through a metamodel described in [154]. Such metamodel proposes three main abstract aspects, which are the basis of the approach:

- *Elements*: any feature or attribute of an SFM, and, in general, any aspect on which users can make decisions. It is denoted by a non-empty set $E = \{E_1, ..., E_n\}$.
- *States*: an SFM has a set of available state sets $AS_1, ..., AS_n$ such that AS_j is the set of available states for an element E_j.
- *Constraints*: an SFM stores both the relationships of the model and the user decisions. It has a set of *relationships* $R = \{R_1, ..., R_i\}$ that sets the conditions a potential configuration must necessarily fulfil to be a configuration of the SFM. It also contains the decisions made by one or more users in a given moment in a set of constraints $U = \{U_1, ..., U_j\}$.

Figure §2.10 shows an example of SFM depicted as a stateful feature diagram. Stateful feature diagrams are a graphical notation of SFMs based on feature diagrams [142]. The representation of elements and relationship constraints is the same that FMs: a

tree-like structure where features are boxes linked by different kinds of lines that represent the relationships among features. Cardinalities are also drawn together with the corresponding set relationship.

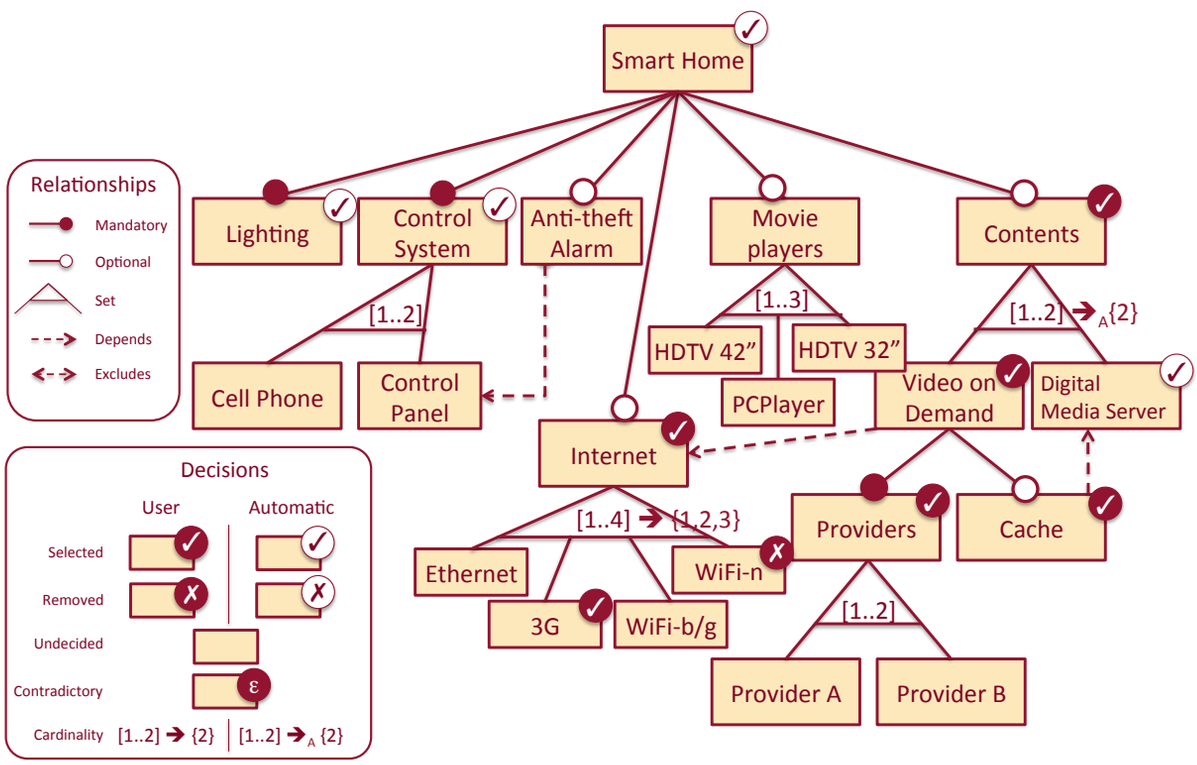


Figure 2.10: An example of a Stateful Feature Diagram

2.3.3 SOUP

Semantic Ontology of User Preferences (SOUP) is a highly expressive, intuitive model of user preferences [65]. A preference can be intuitively expressed as"I prefer y rather than x", where x and y are instances of domain concepts that represent term values such as the OS or desired storage. This relation between concept instances can be mathematically interpreted as a strict partial order.

Figure §2.11 presents a UML representation of the upper ontology of SOUP preference model, where the user basically can express atomic preferences using different preference terms that are handled internally by the corresponding ranking mechanism. Then, composite preferences can be used to compose those terms, defining the relationship between previously expressed atomic preferences. Note that composite preferences are also handled by a ranking mechanism that offers facilities to combine

simpler atomic preferences.

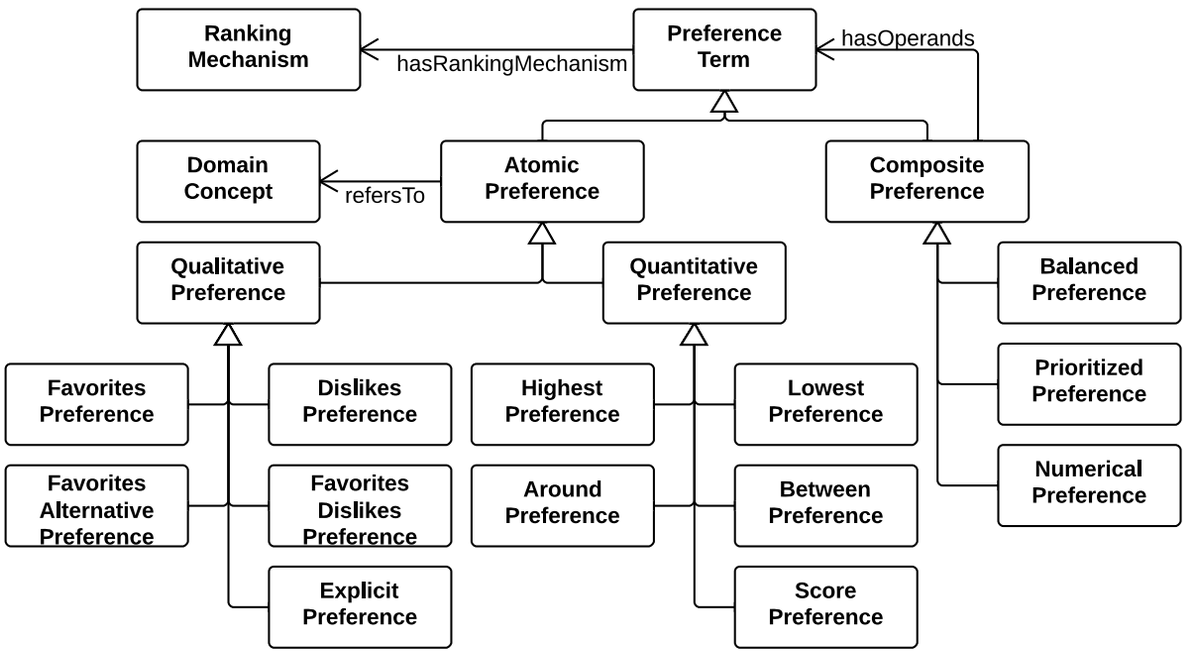


Figure 2.11: SOUP preferences.

In particular, atomic preferences are related to a domain-specific concept that usually represents a non-functional property (NFP) that should be optimized to fulfill the user preference over it. Figure §2.11 outlines the available preference terms from SOUP. In this dissertation we focus on some specific quantitative and qualitative preferences: Favorites, Dislikes, Around, Highest and Lowest. A Favourites (a Dislikes) preference means that the users prefers (dislikes) a specific value for a NFP, A Lowest (a Highest) preference means that the user prefers a lower (higher) NFP value, while an Around preference favours values close to a particular one. The preference model supports other facilities to express additional qualitative, quantitative and composite preferences. A formal description of both atomic and composite preferences of this model model is presented in [64].

The adaptation of this common preference model to a concrete service selection scenario enables interoperability between ranking mechanisms, using it as a preference meta-model, which provides a higher expressiveness compared to each mechanism isolated.

2.4 SUMMARY

From the results of this chapter, we obtain several conclusions that motivate our thesis work presented in this dissertation. First, HCSs are common in the cloud. The growing configuration options and alternatives offered by cloud providers make cloud services difficult to understand and configure – even "simple" services such as Dropbox. It is remarkable that the lower the level in the service model, the higher the number of configuration options and values – closer to IaaS, higher the number of options.

Second, the current approaches to describe the decision space of HCSs present lacks. As far as we know, there are no specific proposals or DSLs for their specification, so their representation mainly relies on natural language and models. While natural language is expressive and comprehensible by most of the consumers, it may be also ambiguous, and hardens a posterior automated processing. On the other side, VM approaches are unambiguous and succinct, and enable automated analysis techniques (as we will see in Section §3.4), but less expressive and comprehensible.

And third, both commercial and research approaches on cloud HCSs neglect user preferences. While the provided expressiveness to make decisions varies from one approach to another, the support to user preferences is null. However, for that aspect we can go to service discovery and ranking area, where there are preference models such as SOUP.

3

HCS-driven Solutions Automated Support

The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency.

Bill Gates (1955),
Businessman

In this chapter, we describe the state of the art for the automation of Highly-configurable Services. Section §3.1 presents the existing support for the search of the best service configuration. In Section §3.2 we describe the particular scenario of the cloud migration, while in Section §3.3 we do the same for the automated reconfiguration of services. Section §3.4 discusses the existing approaches to analyse the variability. Finally, Section §3.5 ends the chapter.

3.1 Search of the Best Configuration

Most of the HCS providers offer web-based apps to configure or simulate a configuration of their services. Amazon provides a calculator for their cloud services [7], including EC2, EBS, S3 and many others (see Figure §3.1). The services can be configured assigning values to their configurable terms – such as OS or instance type for EC2. Other providers, such as Rackspace [130] or Azure [111] also provide this kind of calculators. However, with this tools we cannot search or at least filter configurations for given user needs. Besides, the kinds of decision we can make are simple – selection of particular values for the terms – but not complex logical/arithmetic constraints or preferences.

Additionally, Amazon provides the *Total Cost of Ownership* (TCO) tool to suggest the best configuration given user needs [3]. This tool let users "describe on-premises or colocation configuration to produce a detailed cost comparison with AWS". Although we can specify the value of terms such as the memory, datacenter location or cpu cores, some others are not available – OS for example. Besides, we cannot choose a range of satisfactory values for the terms: the only way to make decisions is selecting a particular value.

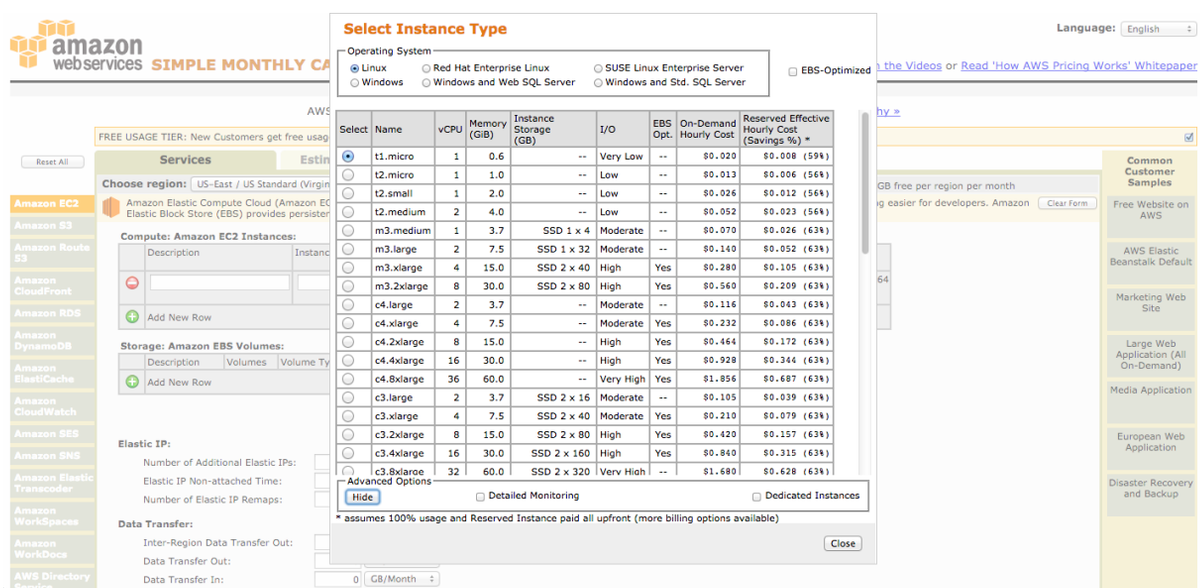


Figure 3.1: Screenshot of the Amazon Calculator.

There are also several web – and independent – applications to assist the search of the most suitable service configuration, mostly on cloud services. CloudScreener [38] let search and compare for cloud infrastructure services among multiple providers.

This application enables the decision making on terms such as cpu, ram or storage for the search. In a similar way, Cloudorado [37] also provides a cloud services price comparison engine. Although both applications enable a higher degree of expressiveness to declare user needs than Amazon TCO, some works have detected false positives in the results – in particular for Cloudscreener [73]. We cannot be sure if they keep updated with the last changes of each provider, since the internal models they use to describe the decision space are not available.

3.2 Cloud Migration

Due to the benefits of cloud environments, during the last years the literature presents significant research on techniques to facilitate the migration of legacy on-premise software to the cloud. Cloud benefits, such cost savings, scalability and on-demand features, make large companies and *Small and Medium Enterprises (SMEs)* want to embrace the cloud. However, this process requires facing a number of issues, such as the need to carry out feasibility studies, provider selection or code/application modifications.

As a proof of the arisen interest, some secondary studies have emerged recently to review the works on cloud migration [86, 95]. In particular, Jamshidi et al. [86] proposes a Cloud Reference Migration Model to address such issues. This reference model encompasses three main phases: planning, execution and evaluation. In particular, this dissertation is concerned about selecting the most suitable provider and its configuration [95] for the migration. For further approaches and related work, we refer to the related work presented in Section §6.9.

3.3 Automated Service Adaptation

The operational environment of services is prone to change. These changes may occur in the required infrastructure resources, for instance due to a partial outage; in the user's side, for instance due to changing needs or users leaving and joining the service; or in other services in which the delivered service relies and which are not under our control. All these cases require a service adaptation in order to make the most of the service under the new circumstances. The kind of adaptation to carry out will depend on the nature of the service and their changes.

For instance, the users of an HCSs and their needs may change at runtime, making

necessary an adaptation process. This seems to make more sense in multi-tenant services, where hardware and software resources and even some configuration capabilities may be shared among multiple users. As an specific example, Wordpress supports multisites, which aggregate several Wordpress [1] [178] sites into a single installation. In this case, the global configuration options, such as the default language, the upgrading policy or the available plugins and themes are shared. Changes in the needs of the users may require to install or remove some plugins or change the upgrading policy.

Service adaptation is a well-known topic in the literature, so several authors have put their focus on it. Cardellini et al. [30] present a reference framework for self-adaptation of service-oriented systems, where the user satisfaction is considered as an adaptation driver. Caton and Rana [31] propose an approach for cloud infrastructure provisioning through volunteered resources, relying on autonomic fault management techniques. In a similar way, Maurer et al. [103] propose an adaptive resource configuration in the cloud for infrastructure management. Nallur and Bahsoon [119] also propose an adaptive mechanism for cloud services, but in this case for applications built on top of different services. The adaptation dynamically selects the best value-for-money services, based on market-based control techniques.

3.4 Automated Support of Variability Models

3.4.1 Automated Analysis of Feature Models

The automated analysis of variability models is about extracting information from FMs using automated mechanisms [18, 19]. Analysing FMs– and VMs in general – is an error-prone and tedious task, and it is infeasible to do manually with large-scale FMs. It is an active area of research and is gaining importance in both practitioners and researchers in the SPL community. Since the introduction of FMs, the literature has contributed with a number of operations of analysis, tools, paradigms and algorithms to support the analysis process.

Benavides et al. [19] describe the most complete catalogue of operations up to now. From this catalogue we select the most representative operations to illustrate the purpose of the AAFM, and its relationship with the configuration process. These operations can be sorted into two main categories: model related operations and configura-

[1] http://codex.wordpress.org/Create_A_Network

tion related operations.

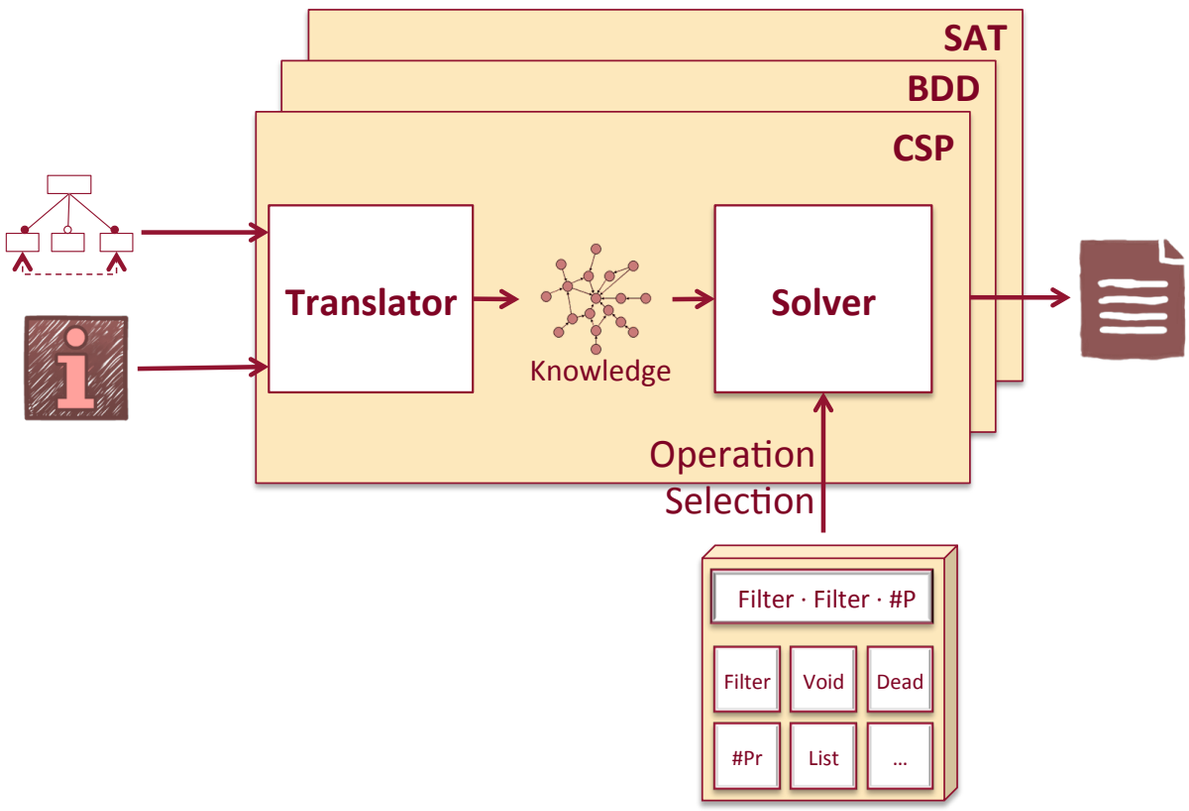


Figure 3.2: Process for the Automated Analysis of Feature Models.

Model-related Operations

- *Void FM*: this operation takes a FM as input and returns a value informing whether such feature model is void or not. A FM is void if it represents no products. The reasons that may make a FM void are related with a wrong usage of cross-tree constraints, i.e. FMs without cross-tree constraints cannot be void.

- *All valid configurations*: this operation takes a FM as input and returns all the valid configurations represented by the model.

- *Number of valid configurations*: this operation returns the number of valid configurations represented by the FM received as input. Note that a FM is void if the number of valid configurations represented by the model is zero.

- *Anomalies detection*: A number of analysis operations address the detection of anomalies in FMs, i.e. undesirable properties such as redundant or contradictory

information. These operations take a feature model as input and return information about the anomalies detected.

  - *Dead features*: A feature is dead if it cannot appear in any of the configurations of the FM. Dead features are caused by a wrong usage of cross-tree constraints.
  - *False optional features*: a feature is false optional if it is included in all the configurations of the FM despite not being modelled as mandatory.
  - *Wrong cardinalities*: a group cardinality is wrong if it cannot be instantiated. These appear in cardinality- based FMs where cross-tree constraints are involved.

- *Explanations*: this operation takes a FM and an analysis operation as inputs and returns information (so-called explanations) about the reasons of why or why not the corresponding response of the operation [155].

Configuration-related operations:

- *Valid configuration*: this operation takes a FM and a configuration (i.e. set of features) as input and returns a value that determines whether the configuration belongs to the set of configurations represented by the FM or not. This operation is sometimes referred as *Valid product* [19].

- *Valid partial configuration*: this operation takes a FM and a partial configuration as input and returns a value informing whether the partial configuration is valid or not, i.e. a partial configuration is valid if it does not include any contradiction.

- *Filter*: this operation takes as input a FM and a partial configuration and returns the set of valid configurations including the input partial configuration that can be derived from the model.

- *Optimisation*: this operation takes a FM and a so-called objective function as inputs and returns the configuration fulfilling the criteria established by the function. An objective function is a function associated with an optimization problem that determines how good a solution is. This operation is chiefly useful when dealing with EFMs where attributes are added to features. In this context, optimization operations may be used to select a set of features maximizing or minimizing the value of a given feature attribute.

Some new operations have been identified after the work of Benavides et al. [19]. This is the case of Henard et al. [80], who propose an operation to fix a FM for supporting a given configuration, or White et al. [175], who present a tailored optimisation operation for spatial deployment.

3.4.2 Automated Analysis of Stateful Feature Models

Recently, Trinidad [154], Trinidad et al. [161] presented a simplified version of the AAFM for the SFMs, the so named *Automated Analysis of Stateful Feature Models (AASFM)*. The AASFM proposes a taxonomy of operations that reduces the operations needed to formalise to a subset of basic operations, while remaining operations are defined on top of them. In this way, the AASFM provides all the AAFM operations and new ones, but relying on a small set of basic operations. Although the AASFM identifies deductive and abductive operations, in this dissertation we focus only on the deductive operations.

AASFM Basic Deductive Operations

- *Products listing*: this operation lists all the products in a SFM that satisfy the current decisions and relationships.
- *Validation*: a validation operation checks if a SFM is valid or not. It means to check if the decisions within the SFM satisfies all the relationships.
- *Propagation*: the propagation operation receives an input SFM and obtains as a result another SFM where the conclusions that are obtained about the states of all the elements are added to the set of automatic decisions.

The rest of operations of the AAFM, except the ones about explanations – which require the abductive operations – can be composed using this three basic ones [154].

3.4.3 Analysis Techniques

The general approach to perform the the automated analysis of both FMs and SFMs is translating the model into a declarative knowledge representation that can be used by existing tools or *solvers* for the extraction of information by automatic means (Figure §3.2). The most used representations are:

- *Propositional logic*: a FM is mapped into a propositional formula that can be evalu-

ated if it is true or false using SAT solvers [23]. Binary Decision Diagrams (BDD) [172] are tree-like structures built from a propositional formula that represent the decisions that can be made by users in a configuration process. SAT and BDD solvers have been used for the AAFM by Batory [15], Czarnecki and Kim [40], Mendonça et al. [108] and Zhang et al. [183].

- *Constraint programming*: a FM is mapped into a *Constraint Satisfaction Problem (CSP)* so a constraint solver can be used to perform different analysis operations. We pioneered this approach in 2005 [18] due to its ability to work with non-boolean attributes. Our efforts in the last years have focused on contributing to the use of constraint programming for the AAFM [157, 158, 174].

- *Description logic*: a FM is mapped into a description logic, which is a representation that describes the knowledge in terms of concepts, roles and individuals [10]. Wang et al. [170] map a FM into a OWL-DL[16], a realisation of description logic in terms of ontologies that are analysed by means of RACER tool [77].

- *Clausal logic*: a FM is maped into a clausal logic. the most used tool to reason on clausal logics [56] is Prolog [85]. Kang [88] proposed using Prolog for some basic AAFM operations 22 years ago, which is considered the first contribution for the AAFM.

- *Others*: a minority of works have proposed other mappings such as Zhang et al. [181, 182] that map FMs into SMV model checker. Some authors have also proposed ad-hoc algorithms to solve specific analysis operations [11, 79, 156, 168, 169].

Each representation enables a subset of analysis operations, each of which having a different performance. So for example a representation can be more suitable for product counting while its performance decreases for product listing – which is the case for BDD.

3.5 SUMMARY

This chapter motivates the need of automated support for HCS-driven solutions. As we have seen, there is a lack of support, from researcher and practitioners, for the search of the best configuration. On the one side, the existing commercial configurators present deficiencies to describe user needs or even false positives in their results. On

the other side, research approaches, in particular in the area of cloud migration, neglect tool support. Thus, we need to rely on other existing disciplines that can provide such automated support. In this sense, the AAFM and their variants – such as the AASFM– offer a catalogue of operations and analysis techniques which seem to be useful for the support of HCSs.

Part III

Contribution

Highly-configurable Services

You never change things by fighting the existing reality. To change something, build a new model that makes the existing model obsolete.

Buckminster Fuller (1895 - 1983),
Architect

In this chapter we present the foundations of HCSs and our proposal for the description of their decision space. Section §4.1 introduces the chapter. In Section §4.2 we define the main concepts of HCSs, while in Section §4.3 we present SYNOPSIS, a DSL for the specification of HCSs. Section §4.4 describes the validity criteria for HCSs in SYNOPSIS notation. In Section §4.5 presents UCL, a SYNOPSIS-based notation to describe user needs on HCSs. Finally, Section §4.6 ends the chapter.

4.1 Introduction

4.2 Abstract Model for Highly-configurable Services

Although the concept of *Highly-configurable Service (HCS)* has been previously employed in the literature, as far as we know there is not a clear definition of what means. Intuitively, we can think of a service with multiple configuration options. However, the difference between a configurable and a highly-configurable service is ambiguous. Consequently, there neither are specific mechanisms to support the description of HCSs and their configuration options.

The purpose of this chapter is to precisely define what an HCSs are, and to provide mechanisms to support the specification of their decision space. For the former goal, first we clearly define our concept of HCSs, including different examples and notations. For the latter one, we propose a notation (SYNOPSIS) to specify the decision space of HCSs– together with validity criteria – and also the user needs on them.

4.2.1 Configurable Services

We consider a service as a resource or functionality delivered by a provider to one or more consumers. The service is described by a number of terms, in which the consumers can have interest. Traditional human-powered services, as electricity or public transport, present terms such as the kWh (kilowatt hour) cost for the former or the transport timetable and the ticket price for the latter. In the same way, web-based services also present terms, such as the provider, the maximum number of requests or the service price.

If some of the service terms have two or more possible values, i.e. they are configurable – or *decision terms* – , we say the service is *configurable*. For example, Dropbox – the storage service – and Spotify – the music streaming service – are both configurable services. These services can be hired with different plans, each providing specific values for the rest of decision terms. In the case of Dropbox the storage limit or the backup options depend on the plan, which may be basic, pro or business. The plan is presented by the provider as a direct choice – or *selectable term* –, while storage and backup options depends on the plan, so they are *derived terms*.

In this context, we define a configurable services as follows:

Definition 4.1 - Configurable Service.
A configurable service CS is a tuple (S, C, D, V, R^S) where S is the set of selectable terms, C is the set of selectable term values, D is the set of derived terms, V is the set of derived terms values and R^S is a set of constraints that describe the dependencies among terms.

The selectable terms are defined in a set $S = \{S_1, ..., S_n\}$ such that each element $S_i \in S$ is linked to a set of term values $C_i \in C$ where $C_i = \{c_{i,1}, ..., c_{i,j}\}$. A user must choose one and only one term value for each selectable term to define a *configuration*. This way, the set of all the potential service configurations can be defined as the cartesian product $C = C_1 \times ... \times C_n$. So for example in Dropbox, a user can only choose a plan among three options: basic, pro and business. We can represent this scenario as follows:

$$S = \{plan\}$$
$$C = \{C_{plan}\}, \text{ s.t. } C_{plan} = \{basic, pro, business\}$$

The derived terms that affect a configurable service can be modelled as a set $D = \{D_1, ..., D_m\}$ where each $D_i \in D$ can take any value in a set $V_i = \{v_{i,1}, ..., v_{i,k}\}$. Following with the Dropbox example, the cost and maximum storage are derived terms that are not configurable but their value changes depending on the chosen plan. They can be modelled as follows:

$$\begin{aligned} D &= \{cost, storage\} \\ V_{cost} &= \{c \in \mathbb{R} | c \geq 0\} \\ V_{storage} &= \{s \in \mathbb{N} | s > 0\} \\ V &= V_{cost} \times V_{storage} \end{aligned}$$

With C and V defining all the values that selectable and derived terms can take, $C \times V$ defines the space of values that terms in a configurable service can take. However, not any combination of them is allowed and values can be bound in different forms, affecting to the way a service can be configured. For example, cost and storage values depend on the plan a user chooses. In order to represent such dependencies, we define a set of constraints $R^S = \{R^S_1, ..., R^S_m\}$ in the space $C \times V$ that define the decision space as follows:

Definition 4.2 - Decision space.
Let $R^S = \{R_1^S \wedge ... \wedge R_m^S\}$ be a set of constraints on $C \times V$. The decision space (*dspace*) of a configurable service is defined as:

$$dspace(CS) \quad = \{(c_1,...,c_n,v_1,...,v_n) \in C \times V | R_1^S \wedge ... \wedge R_m^S\}$$

The decision space of the Dropbox service can be defined adding the following constraints to the configurable service description:

$$\begin{aligned} R_1^S &= \{c_1 = basic \Rightarrow v_{cost} = 0.00 \wedge v_{storage} = 5\} \\ R_2^S &= \{c_1 = pro \Rightarrow v_{cost} = 5.00 \wedge v_{storage} = 1000\} \\ R_3^S &= \{c_1 = business \Rightarrow v_{cost} = 10.00 \wedge v_{storage} = \infty\} \end{aligned}$$

These constraints generate the following decision space:

$$\begin{aligned} dspace(CS_{Dropbox}) = \{ \quad & \{basic, 0.00, 5\}, \\ & \{pro, 5.00, 1000\}, \\ & \{business, 10.00, \infty\}\} \end{aligned}$$

In order to be able to refer to the set of configurations of a configurable service independently of the values of derived terms, we define a configuration space as follows:

Definition 4.3 - Configuration space.
Let $proj_C : C \times V \rightarrow C$ be a projection function that binds each element in the decision space to a configuration in C. The configuration space of a configurable service is defined as:

$$cspace(CS) \equiv proj_C(dspace(CS))$$

The configuration space of Dropbox – which in this case coincides with the potential set of configurations C – is $\{basic, pro, business\}$.

4.2.2 User Configurations

A configurable service requires consumers to choose a configuration in order to be delivered. In most of the cases, the consumer only has to assign a value to a subset of

these terms, the selectable ones. In the case of Dropbox the plan determines the configuration of the service; or in Amazon EC2, the region, instance type, OS, purchasing plan, dedication and extra storage are the selectable terms to be decided. The remaining terms depend on them, but are an important source of information in order to make a decision about the configuration that best suits the consumer needs. So for example a consumer may make a decision about Dropbox based on the cost and the storage.

The consumer needs can be defined as a set of constraints $R^U = \{R_1^U \wedge ... \wedge R_k^U\}$ that helps to reduce the decision space to those values that satisfy the provided constraints. We define the filtered decision space as follows:

Definition 4.4 - Filtered decision space.
Let R^U be a set of constraints on the decision space that represents a set of consumer needs. The filtered configuration ($filter$) of a given service $CS = (S, C, D, V, R)$ and set of requirements R^U is defined as:

$$\begin{aligned} filter(CS, R^U) &= \{(c_1, ..., c_n, v_1, ..., v_n) \in dspace(CS) | R^U\} \\ &\equiv \{(c_1, ..., c_n, v_1, ..., v_n) \in C \times V | R \wedge R^U\} \end{aligned}$$

For example, if a consumer needs a storage service that costs less than 6$ a month, and provides more than 20GB of storage, the user needs constraints can be defined as follows:

$$\begin{aligned} R_1^U &= \{v_{cost} < 6.00\} \\ R_2^U &= \{v_{storage} > 20\} \end{aligned}$$

In this case, the filtered decision space just contained one tuple in the form $(pro, 5, 1000)$, that corresponds to a service configuration $\{pro\}$.

4.2.3 Highly-Configurable Services

The configuration capabilities of a service may go further, with the contract of multiple items – a.k.a. instances – of the service that may have different configurations, or even additional linked services. This leads to the so-named Highly-configurable Services. While some configurable services do not allow this – e.g. Dropbox or Spotify – others do, such as EC2 or Heroku. In the case of EC2, we can contract different computing instances of different types and in different regions, and even additional storage through the EBS service, all of them related to the same Amazon account. In the case of

Heroku, we can also contract different Dynos and Postgres of different types. Different items of the same service may be interrelated by means of dependencies. For instance, Amazon – and other similar providers such as Rackspace – provides a volume-based discount which depends on the total cost of all the items contracted.

We propose the following definition to capture the particularities of HCSs:

Definition 4.5 - Highly-configurable service.
Let $CS = S_1,...,S_k$ be a set of configurable services; let $T = \{T_1,...,T_o\}$ be a set of HCS-specific terms whose values are defined in $V = \{V_1,...,V_o\}$; let $I = \{I_1,...,I_k\}$ be a set of service items such that each $I_i \in I$ is a set $I_i = \{R^U_{i,1},...,R^U_{i,n}\}$ of consumer constraints that define the requirements for each item of the configurable service CS_i; let N be a set of invariants in I that limits the number of items for each service; and let R_M be a set of constraints among instances in the set I.

$$HCS = (CS, T, V, I, N, R^M)$$

If we consider Amazon as an HCS that permits the configuration of multiple instances of EC2 and EBS, having a 10% discount for orders greater than 1,000 $, it could be modelled as follows:

$$\begin{aligned}
CS &= \{\ CS_{ec2}, CS_{ebs}\} \\
T &= \{\ total, subtotal\} \\
V &= \{\ \mathbb{R}, \mathbb{R}\} \\
I &= \{\ \emptyset, \emptyset\} \\
N &= \{\ |I_{ebs}| > 0 \Rightarrow |I_{ec2}| > 0\} \\
R^M &= \{\ subtotal = \textstyle\sum_{ij} cost_{ij}, \\
&\qquad subtotal > 1{,}000 \Rightarrow total = 0.9 \cdot subtotal, \\
&\qquad subtotal \leq 1{,}000 \Rightarrow total = subtotal\}
\end{aligned}$$

In an HCS the consumer needs are expressed by means of the I set, where each configurable service that comprises an HCS can be configured as many times as needed, on the condition that the invariants in N must be always satisfied.

4.2.4 Conceptual Metamodel

Figure §4.1 shows a conceptual model that organises and binds all the concepts presented in this Section. A service is described by a set of terms. In the case of a

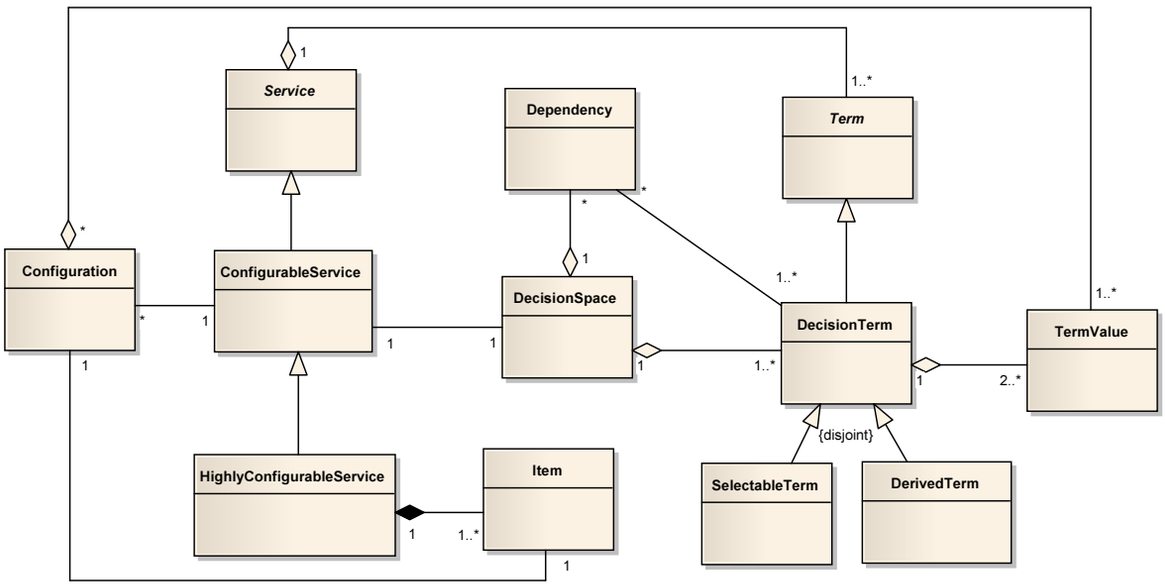


Figure 4.1: Highly-configurable Service Conceptual Metamodel.

configurable service, some of these terms are configurable terms with at least two possible values. If the provider offer different alternatives for a configurable term, we say the term is selectable – and derived in other case. The different configurable terms and their dependencies makes up the so-named decision space of the service. Such decision space encompasses all the available configurations – valid combinations of configurable term values. In the case a configurable service let hire multiple and configurable items – or instances –, we say the service is a Highly-configurable Service. Each of the service items can have different configurations.

4.3 SYNOPSIS

SYNOPSIS (SimplY a NOtation to sPecify Service configuratIonS) is a text-based, human-readable notation to describe the decision space of services. It supports the specification of the configuration capabilities described in Section §4.2, common to the services of big providers such as Amazon, Rackspace or Microsoft, while remaining provider-agnostic. Figure §4.2 shows the SYNOPSIS description of a simple block storage service, which we will use as running example to explain notation.

Figure 4.2: Simple Block Storage Service in SYNOPSIS.

```
Service SimpleBlockStorage{

        %Terms
        ## Selectable terms
        SSD: boolean;
        Size: int [1,1000];
        Region: {"USA", "EU", "JP"};
        ## Derived terms
        costGBMonth: real [0.00,0.15]; ## euros/GB per month
        volumeCostMonth: real [0.00,150]; ## euros per month

        %Dependencies
        ## pricing
        TABLE
          Region       SSD     -> costGBMonth;
            "USA"     true     ->    0.1;
            "EU"      true     ->    0.12;
            "JP"      true     ->    0.15;
            "USA"     false    ->    0.05;
            "EU"      false    ->    0.06;
            "JP"      false    ->    0.08;
        ENDTABLE

        volumeCostMonth == costGBMonth * Size;
}
```

4.3.1 Configurable and Highly-configurable Services

SYNOPSIS notation let define the decision space of configurable services and HCSs– the latter is an aggregation of the former. A SYNOPSIS document starts with the definition of the service type – `Service` keyword for configurable services, and `Highly-configurable Service` for HCSs. While Figure §4.2 shows a configurable storage service, Figure §4.3 shows an aggregation of block storages, giving rise to a storage HCSs.

A configurable service in SYNOPSIS has two sections, to declare terms and dependencies. In the terms section we declare the different terms – both selectable and derived – and their values, while in the dependencies section we describe the relationships among the different terms.

An HCS in SYNOPSIS has three sections to declare the component services and

Figure 4.3: Volume Storage HCS in SYNOPSIS.

```
Highly-configurable Service VolumeStorage{                                  1
                                                                            2
        %Services                                                           3
        SimpleBlockStorage[1,*]  storage;                                   4
                                                                            5
        %Terms                                                              6
        totalCostMonth: real [0.00,10000.00];                               7
        discount: real [0.00,1000.00];                                      8
                                                                            9
        %Dependencies                                                       10
        ## cost aggregation                                                 11
        totalCostMonth == SUM(storage.volumeCostMonth);                     12
        ## discount policy                                                  13
        totalCostMonth > 3000 -> discount == totalCostMonth*0.1;            14
}                                                                           15
```

their cardinality, global terms and global dependencies. For the two latter sections, the syntax is the same than for configurable services, and is describe in the following subsections. In the `%Services` section we declare the service that compose the HCS, their cardinality – lower and upper bounds for the items – and an alias to be referred.

4.3.2 Decision Terms

Decision terms are declared in SYNOPSIS in the terms section, denoted by the tag `%Terms`. As shown in Section §4.2, terms can be classified into two disjoint categories: selectable terms and derived terms. In this sense, both term types are described in this section and treated in the same way. The only requirements is that each term should have at least two different term values.

The declaration syntax in shown in Figure §4.2. First, we define the term's name, and after a colon we declare its domain. Such domain can be enumerated, boolean, integer or real. In the case it is enumerated, the values are surrounded by quotation marks, separated by commas and enclosed by brackets. In the case of integer or real values, we have to use the keyword `int` or `real` and define the domain by means of the upper and lower bounds. For real domains, we can specify the float using as positions as necessary in the declaration.

In Figure §4.2, we can see the four different types of decision terms. We have also

used comments to separate selectable terms and derived terms. For example, `SSD` is a boolean selectable term that indicates if the storage is ssd-based, `Size` is an integer selectable term that indicates the storage capacity, `Region` is an enumerated selectable term that declares the available regions of the service, and `costGBMonth` is a real derived term to describe the cost hour of the different configurations.

HCS Terms

As we have said before, an HCS can define global terms that affect the whole aggregation of its configurable services This kind of term is necessary to describe, for instance, the total cost or the discount of an HCS, which depends on all the aggregated services. These terms are declared the terms section of the HCS in the same way than the rest of terms. Besides the standard operators for dependencies, order terms have available especial types – described in Section §4.3.3.

Figure §4.2 shows a couple of HCS terms: the total cost and the discount. The value of `totalCostMonth` is calculated based on the aggregation of the `volumeCostMonth` of each storage item, while the `discount` is calculated as a 10% of the total cost when it exceeds 3000 euros.

4.3.3 Dependencies

SYNOPSIS provides a set of expressions and operators in order to define the dependencies in the decision space. They include the classic logical, relational and arithmetic operators, and also aggregation functions to relate HCS terms to standard terms. There are four main expression types: logical, integer, real and enumerated. Every dependency declared in SYNOPSIS should be logical, although it may be composed by other expression types. Table §4.1 summarises such expressions [1] [2]

Additionally, SYNOPSIS provides tables to declare groups of dependencies which involve the same terms with different values. Figure §4.2 and Figure §4.4 show some examples of these tables, which are useful to describe, for instance, the characteristics of given values of a term – as the case of computing instances – or the pricing policies. The way the work is simple: in the first row, we declare the header of the table, i.e. the configurable terms, and the dependency relationship – from the left to the right, split by the implication -> symbol. Each additional row provides the values for each term.

[1] $\mathbb{E}$ represents the set of all enumerated values of the document.
[2] *sum*, *min* and *max* functions aggregate standard terms into order terms.

Dependency expressions	
Type	**Expressions**
Boolean	$B ::= b \mid t_b \mid B\&\&B \mid B \parallel B \mid !B \mid B\text{->} B \mid B \text{<->} B \mid I > I \mid I >= I \mid I < I \mid I <= I \mid I == I \mid I! = I \mid R > R \mid R >= R \mid R < R \mid R <= R \mid R == R \mid R! = R \mid E == E \mid E! = E$
Integer	$I ::= i \mid t_i \mid I + I \mid I - I \mid I * I \mid I/I \mid -I \mid I^I \mid sum(t_i) \mid max(t_i) \mid min(t_i)$
Real	$R ::= r \mid t_r \mid R + R \mid R - R \mid R * R \mid R/R \mid -R \mid sum(t_r) \mid max(t_r) \mid min(t_r)$
Enumerated	$E ::= e \mid t_e$

t_b any boolean term, t_i any integer term, t_r any real term, t_e any enumerated term.
$b \in \{true, false\}, i \in \mathbb{Z}, r \in \mathbb{R}, e \in \mathbb{E}$

Table 4.1: Expression Types for SYNOPSIS

4.4 Validity Criteria

A configurable service may present different anomalies regarding its configuration capabilities. For example, it is possible that some values of a configurable term are no selectable under any circumstance, or that a configurable term is not such configurable. In this section, we define the validity criteria for configurable services, which is composed by three levels and five anomaly types. In particular, the levels are as follows:

1. **Warning level**: this level encompasses anomalies that do not damage the configuration capabilities of the service.
2. **Term error level**: this level encompasses anomalies that damage the configurations capabilities of the service, and in particular of given values and terms.
3. **Service error level**: the errors of this level makes the service no configurable or directly inconsistent.

In order to illustrate the anomalies, we use in the following a running example of a Simple Computing Service defined in SYNOPSIS notation (Figure §4.4). This example is a simplified version of typical computing services, such as Amazon EC2 or

Rackspace Servers. We have specified three different configurable terms – instance type, OS and Region – and the memory, virtual CPU and cost of the service. We also describe their characteristics and pricing policy by means of two tables, and additional constraints.

Figure 4.4: Simple Computing Service in SYNOPSIS.

```
Service SimpleComputingService{

        %Terms
        ## Selectable terms
        InstanceType: {"S", "M", "L", "XL"};
        OS: {"Debian","Windows"};
        Region: {"USA", "EU", "JP"};
        ## Derived terms
        InstanceMemory: int [1,8]; ## GB
        InstancevCPU: int [1,4]; ## cores
        costHour: real [0.0,1.0]; ## euros/hour

        %Dependencies
        Region == "JP" -> InstanceType != "XL";
        InstanceType == "Windows" -> InstanceMemory >= 2

        TABLE ## instance characteristics
        InstanceType -> InstanceMemory InstancevCPU;
        "S"     ->      1               1;
        "M"     ->      2               2;
        "L"     ->      4               3;
        "XL"    ->      6               4;
        ENDTABLE

        TABLE ## pricing
        InstanceType OS         -> costHour;
            "S"     "Debian"    ->    0.1;
            "M"     "Debian"    ->    0.2;
            "L"     "Debian"    ->    0.4;
            "XL"     "Debian"    ->    0.8;
            "S"     "Windows"   ->    0.15;
            "M"     "Windows"   ->    0.3;
            "L"     "Windows"   ->    0.5;
            "XL"     "Windows"   ->     1;
        ENDTABLE
}
```

4.4.1 Warning Level

We name this first validity level as the warning level. As the name denotes, the warning of this category does not affect the configuration capabilities of the service, but may harden the understanding of the decision space. In particular, we have identified one anomaly in the warning level: the redundant dependency.

A redundant dependency has no effect on the decision space of the service. If such dependency is removed, the resultant decision space remains unaltered, what can be defined as follows;

Definition 4.6 - Redundant dependency.

$$redundant((S,C,D,V,R^S),R_i^S) \equiv dspace(S,C,D,V,R^S) = dspace(S,C,D,V,R^S - \{R_i^S\})$$

In Figure §4.5 we see an example of redundant decision space for the computing service of Figure §4.4. The dependency says "if instance is XL, the instance memory should be greater than 5", while at the same time we say in the first table of Figure §4.4 that the memory of an XL instance is 6. In this way, such dependency does not modify the decision space, and can be classified as redundant.

Figure 4.5: Example of Redundant Dependency.

```
...                                                                    1
## a redundant dependency                                              2
InstanceType == "XL" -> InstanceMemory > 5;                            3
...                                                                    4
```

4.4.2 Term Error Level

We name this second validity level as the term error level. Although in this level the service still presents multiple configurations, these two errors damage its configuration capabilities. We identify two types of errors that affect single values and terms: dead values and false decision terms.

A dead value in a selectable term is a value which cannot be selected under any circumstances. In this way, although the value can be apparently chosen, existing dependencies make it non selectable. A dead value can be defined as follows:

Definition 4.7 - Dead value.
A value of a selectable term is dead if there is no configuration in the decision space where that value can be chosen:

$$dead(CS, C_{i,j}) \equiv \nexists(c_1,...,c_n,v_1,...,v_m) \in dspace(CS) \cdot c_i = C_{i,j}$$

In Figure §4.6 we see an example of a dead value for the computing service of Figure §4.4. Since there is no any instance type whose memory is more than 5 GB, the ``Debian'' value for the OS selectable term cannot be selected under any circumstance and consequently is dead.

Figure 4.6: Example of Dead Value and False Decision Term.

```
...                                                    1
## a constraint that generates a dead value            2
## and a false configurable term                       3
OS == "Debian" -> InstanceMemory > 5;                  4
...                                                    5
```

If all the term values but one of a given decision term are dead, we say the term is a false decision term. Although the term shows an appearance of configurable, there is no possible decision: the consumer is forced to select a single value. We define false decision terms as follows:

Definition 4.8 - False decision term.
A decision term is false if it must be chosen in every configuration in the decision space. A false decision term makes all the remaining alternatives for its configuration option to be dead.

$$falseDecision(CS, C_{i,j}) \equiv \forall(c_1,...,c_n,v_1,...,v_m) \in dspace(CS) \cdot c_i = C_{i,j}$$

In the same Figure §4.6 we also see an example of a false decision term for the computing service of Figure §4.4. Given that the term OS only has two values, the dead of one of them makes the term a false decision term. In this case, the consumer cannot choose among two OS: she has to select ``Windows'' always.

4.4.3 Service Error Level

We name this third validity level as the service error level. In this level, the service presents one or none configurations, so consequently these errors are the most critical ones. We identify two types of service errors: false configurable service and inconsistent service.

A service is a false configurable service when there is only a single available configuration. In other words, all the decision terms of a false configurable service are false decision terms, i.e. there is no real choices:

Definition 4.9 - False configurable service.
A service is false configurable if its decision space contains only one possible configuration.

$$falseConfigurable(CS) \equiv |cspace(CS)| = 1$$

In Figure §4.7 we see a set of dependencies that make the service of Figure §4.4 a false configurable service. In this way, only one configuration can be selected: `OS == ``Windows'', Region == ``EU'', InstanceType = ``S''.`

Figure 4.7: Example of False Configurable Service.

```
...
## constraints that makes the service
## non-configurable (only 1 configuration)
OS == "Debian" -> InstanceMemory > 5;
Region == "USA" -> OS == "Debian";
!Region=="JP";
Region =="EU" -> costHour < 0.3;
...
```

In the case all the values of a decision term are dead, we say that the service is an inconsistent service. This means that there is no available configuration for the service, and consequently it cannot be delivered to the consumer. We define inconsistent configurable services as follows:

Definition 4.10 - Inconsistent configurable service.
A configurable service is inconsistent if its decision space is empty.

$$inconsistent(CS) \equiv dspace(CS) = \emptyset$$

In Figure §4.8 we see the dependencies of the previous examples with an additional one, which makes the service of Figure §4.4 inconsistent. There is no available configuration which satisfies all these dependencies.

Figure 4.8: Example of an Inconsistent Service.

```
...
## constraints that makes the service
## non-configurable (only 1 configuration)
OS == "Debian" -> InstanceMemory > 5;
Region == "USA" -> OS == "Debian";
!Region=="JP";
Region =="EU" -> costHour < 0.3;
OS == "Windows" -> Region != "EU";
...
```

4.4.4 Discussion

All the aforementioned errors but the warnings can be interpreted in terms of a single, basic error type: the dead value. Particular combinations of dead values over one or more terms lead to the rest of errors. In this sense, a false decision term is a term whose all its values but one are dead. A false configurable service is a service whose all its terms are false decision terms. i.e. all the values but one of all these terms are dead. And an inconsistent term is a service where one or more terms have not available values, i.e. all the values of such term/s are dead.

Furthermore, variants of the presented errors may appear depending on the consumer. For example, there may be conditional dead values, which are restricted to particular consumers. This is the case of Amazon EC2 and the GovCloud [3] area, which is "an isolated Region designed to allow US government agencies and customers to move sensitive workloads into the cloud".

[3] http://aws.amazon.com/es/govcloud-us/

4.5 User Configuration Language

We propose the *User Configuration Language (UCL)*, a notation based on SYNOPSIS to describe the needs of the consumers of a service. Figure §4.9 presents an example that includes most of the constructions of the language. While SYNOPSIS is oriented for providers to declare the decision space of a service, UCL is oriented to consumers to express what they need on the service. In a UCL document, a consumer can express needs in terms of service items, requirements and preferences. In the following, we describe these three aspects.

Figure 4.9: User needs on the Simple Block Storage Service.

```
Needs on VolumeStorage{

        %Items
        storage["vol1","vol2"];

        %Requirements
        storage["vol1"].Size == 500;
        storage["vol1"].Region == "USA";
        storage["vol2"].Size >= 200;


        %Preferences
        Favorites(storage["vol2"].SSD);
        Dislikes(storage["vol2"].Region, "JP");
        Lowest(SimpleBlockStorage.totalCostMonth);
}
```

4.5.1 Service and Items

In the header of a UCL file, we declare on which configurable service or HCS we declare our needs. In the case of an HCS, we must define in section `%Items` how many items we want of each component service, together with their alias, using an associative array syntax. After that, we use these aliases to express our needs on the services. For instance, in Figure §4.9 we declare two SimpleBlockStorage items, `vol1` and `vol2`.

4.5.2 User Requirements

In the second section of a UCL document, we declare our requirements on the previously defined items. A requirement can be defined as a constraint which must be satisfied by a given configuration. The available operators and expressions to define requirements are the same than for the Dependencies section of SYNOPSIS (see Table §4.1). In Figure §4.9 we can see three requirements for the two items previously declared. We employ an object-oriented notation, where a dot separates the alias of the service item (left side) from the term (right side).

4.5.3 User Preferences

In the third and last section of a UCL document, we declare our preferences on the items. These preferences are a subset of the SOUP user preferences [65], employed for the ranking of services. For our case, we have adapted five SOUP preferences in order to describe fuzzy user preferences on the configurable terms of a service. We employ a prefix syntax, where the preference operator receives one or two arguments depending on the specific preference. To refer to the service items, we employ the previously defined aliases. The preference operators are as follows:

- *Favorites* defines a boolean or enumerated term value desired by the user. It receives the term and specific value as inputs – in the case of a boolean term, only the term is required. For example, `Favorites(storage["vol1"].Region,"US")`.
- *Dislikes* defines a boolean or enumerated term value not desired by the user. It receives the term and specific value as inputs – in the case of a boolean term, only the term is required. For example, `Dislikes(storage["vol1"].SSD)`.
- *Highest* defines a preference on the highest possible value for a given real or integer term. It receives the term as input. For example, `Highest(storage["vol1"].Size)`.
- *Lowest* defines a preference on the lowest possible value for a given real or integer term. It receives the term as input. For example, `Lowest(SimpleBlockStorage.totalCostMonth)`.
- *Around* defines a preference on a real or integer term to be around a specific value defined by the user. It receives the term and the specific value as inputs. For example, `Around(storage["vol1"].costGBMonth,0.1)`.

In Figure §4.9 we can see some user preferences. While the two first preferences are expressed on item level terms, the last one refers to an HCS term. For that case, we employ the name of the service – as the static attributes in java – as alias.

4.6 SUMMARY

In this chapter, we have provided intuitive and formal definitions of what a configurable and Highly-configurable services are. As far as we know, this is the first time that a precise definition of a HCS is proposed. We have also proposed a set of validity criteria to determine if HCSs present anomalies in their configuration space, or even if they are such configurable.

The lacks found in the literature has motivated us to propose SYNOPSIS, a DSL to describe the decision space of HCSs. SYNOPSIS makes possible the textual specification of configurable services and their composition – HCSs. This specification is heavily inspired in textual variability languages, and in particular in TVL (see Section §2.2). Additionally, we have defined UCL, a DSL for the description of user needs which takes inspiration from the syntax and structure of SYNOPSIS. In UCL, users can define the items, requirements and preferences they need on HCSs.

HCS Automated Analysis

He who would search for pearls must dive below.

John Dryden (1631 - 1700),
Poet

In this chapter, we detail our approach for the automated analysis of HCSs. In Section §5.1 we introduce the chapter. Section §5.2 describes the mapping of HCSs to SFMs. In Section §5.3 we present the analysis operations for configurable, while in Section §5.4 we do the same for the HCSs. Finally, a brief summary is presented in Section §5.5.

5.1 Introduction

Although SYNOPSIS provides a way to specify HCSs and a set of validity criteria, analysis techniques are still required to enable automated support. For this aim, we need first to provide SYNOPSIS with formal semantics. In this sense, if the target domain of the formalisation enables analysis operations, we could compose the operations of HCSs on top of them. For this reasons, we choose SFMs as the target domain: we can take advantage of the AASFM operations for the analysis of HCSs.

After the formalisation, we should design a set of analysis operations to, at least, 1) automate the checking of the validity criteria and 2) assist consumers in their decision making on HCSs. Each validity criterion leads to an analysis operation, while users may require several operations for the decision making, e.g. to check if their requirements on a HCS are feasible or to search for the best configuration. Additionally, more analysis operations can make sense for explaining detected anomalies or for the verification of the decision space.

5.2 Formal Semantics

5.2.1 Primary Goal

Our primary formalisation goal is to provide HCSs with automated analysis techniques. Therefore, according to Hofstede and Proper [82], we should choose the formalisation style accordingly – *Primary goal principle*. Our choice is to follow a translational style, i.e. give the semantics to HCSs by the definition of a mapping to another model – a *target domain*. In this case, we assign formal semantics to HCSs through a mapping to SFMs, presented in Section §2.3.

The main strong point of mapping HCSs to SFMs is taking advantage of the analysis operations of the AASFM. SFMs are intended to describe all the possible configurations of variability-intensive systems and assist their configuration, while providing a catalogue of analysis operations. As shown in Figure §5.1, the translation of HCSs to SFMs enables the use of the AASFM analysis operations, so we can build the HCSs analysis operations on top of them.

Hofstede and Proper [82] also state with the *semantic priority principle* that the formalisation focus should be on semantics, not on syntax. Therefore, for a translational

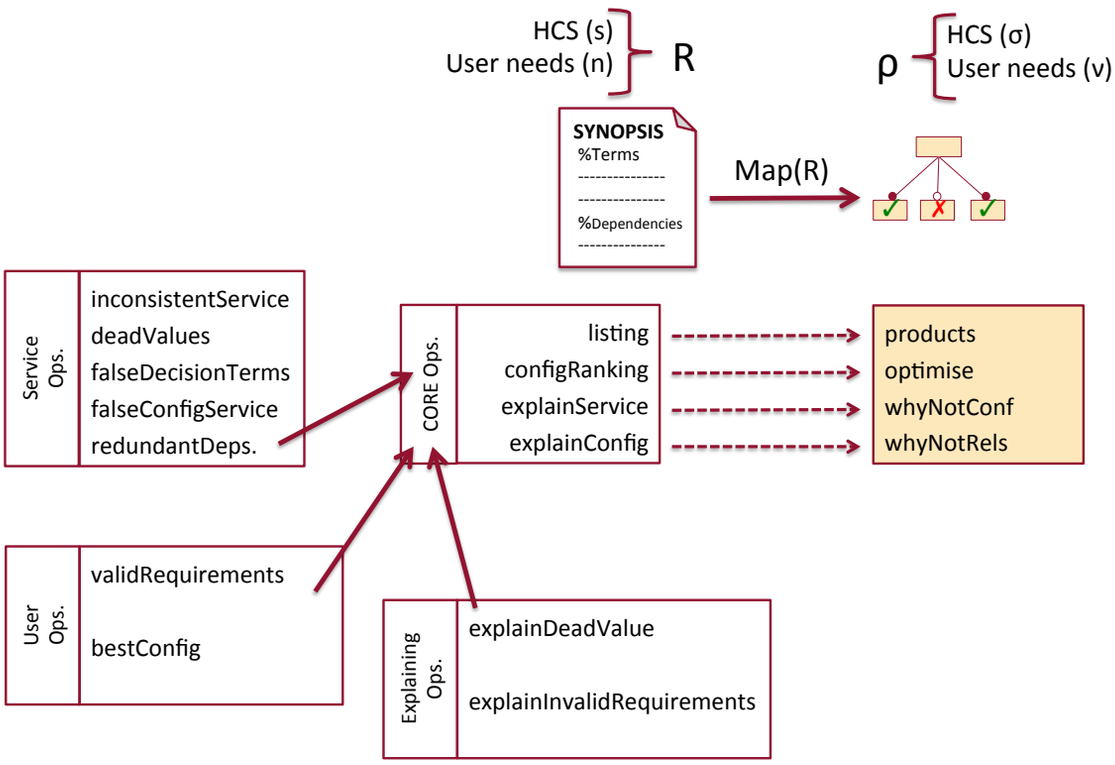


Figure 5.1: Overview of our approach for the automated analysis of HCSS.

semantics style, the semantic distance between the two models should be short. This also makes SFM suitable as target domain, since SFMs are employed for the description and configuration of variability, in a similar way than our purpose of describing and configuring HCSs.

5.2.2 Mapping CSs to SFMs

Let us assume that we have an algorithm $map : CS \times R^U \mapsto SFM$ that generates an SFM from a given configurable service. This algorithm creates an abstract feature for each selectable term in the HCS, linking them together with an abstract root feature by means of mandatory relationships. Every term value for each selectable term also corresponds to a leaf feature, that is linked with the parent feature that corresponds to its selectable term. So for example, the Dropbox example selectable terms generate the following elements and relationships as a result of the mapping:

$$\begin{aligned} S = &\ \{plan\} \\ C = &\ \{C_{plan}\}, \text{ s.t. } C_{plan} = \{basic, pro, business\} \end{aligned}$$

$$\begin{aligned} E \times D = & \{F_{Root}, F_{plan}, F_{basic}, F_{pro}, F_{business} : \{sel, rem\}\} \\ R = & \{mandatory(F_{Root}, F_{plan}), alternative(F_{plan}, F_{basic}, F_{pro}, F_{business})\} \end{aligned}$$

Derived terms are mapped into attributes whose respective domains are defined by the set of term values. The derived terms in the Dropbox example are:

$$\begin{aligned} D &= \{cost, storage\} \\ V &= \{\{c \in \mathbb{R} | c \geq 0\}, \{s \in \mathbb{N} | s > 0\}\} \end{aligned}$$

In this example, two attributes are created, one for each derived term. The domain is taken from the set of values and the constraints that reduce them are mapped into relationships on their respective attributes:

$$\begin{aligned} E \times D &= ... \cup \{F_{Root}.cost : \mathbb{R}, F_{Root}.storage : \mathbb{N}\} \\ R &= ... \cup \{F_{Root}.cost \geq 0, F_{Root}.storage > 0\} \end{aligned}$$

The dependencies among terms defined in the set of constraints R^S are mapped into relationships in the same form that refer to the corresponding feature and/or attributes instead of the terms. The following constraints are defined in the Dropbox example:

$$\begin{aligned} R_1^S &= \{c_1 = basic \Rightarrow v_{cost} = 0.00 \wedge v_{storage} = 5\} \\ R_2^S &= \{c_1 = pro \Rightarrow v_{cost} = 5.00 \wedge v_{storage} = 1000\} \\ R_3^S &= \{c_1 = business \Rightarrow v_{cost} = 10.00 \wedge v_{storage} = \infty\} \end{aligned}$$

In order to keep track of the relationships between terms in the HCS and elements in the SFM, a traceability table is built across the mapping process. Table §5.1 presents the traceability table for this example. The following relationships are obtained when terms in R^S are substituted by the corresponding elements in the SFM:

$$\begin{aligned} R = ... \cup \{ & F_{basic} = sel \Rightarrow F_{Root}.cost = 0 \wedge F_{Root}.storage = 5, \\ & F_{pro} = sel \Rightarrow F_{Root}.cost = 5 \wedge F_{Root}.storage = 1000, \\ & F_{business} = sel \Rightarrow F_{Root}..cost = 10 \wedge F_{Root}.storage = \infty\} \end{aligned}$$

User needs describe the requirements and preferences. For the Dropbox example we have the following requirements:

$$\begin{aligned} R_1^U &= \{v_{cost} < 6.00\} \\ R_2^U &= \{v_{storage} > 20\} \end{aligned}$$

Dropbox Traceability Table	
Selectable Term	**Feature**
$plan \in S$	F_{plan}
$basic \in C_{plan}$	F_{basic}
$pro \in C_{plan}$	F_{pro}
$business \in C_{plan}$	$F_{business}$
Derived Term	**Attribute**
$cost \in D$	$F_{Root}.cost$
$storage \in D$	$F_{Root}.storage$

Table 5.1: Traceability table between HCS and SFM elements for the Dropbox example

Every user need in R^U is mapped onto SFMs the same manner than the service constraints, but they are added to the user decisions structure U in the SFM.

$$U = \{F_{Root}.cost < 6, F_{Root}.storage > 20\}$$

In order to take user preferences into account, each element in the SFM is assigned a p attribute that takes values in $[0,1]$ that remarks the interest of a user in a feature or attribute. $p = 1$ indicates the highest possible interest, while $p = 0$ indicates the lowest. The overall interest in a configuration is reflexed by the global preference attribute $F_{Root}.p$, which is the average value of all the p attributes in the SFM. So one configuration is preferred to another when the p value of the first is greater that the p value of the second.

$$\begin{aligned} E \times D = \{ \quad & F_{Root}.p, F_{basic}.p, F_{pro}.p, F_{business}.p : [0,1], \\ & F_{Root}.p_{cost}, F_{Root}.p_{storage} : [0,1]\} \\ R = ... \cup \{ \quad & F_{Root}.p = (F_{basic}.p + F_{pro}.p + F_{business}.p + F_{Root}.p_{cost} + F_{Root}.p_{storage})/5\} \end{aligned}$$

This way, we create a simple preference structure on top of which user preferences can be defined in quantifiable terms. Table §5.2 shows how preferences are mapped onto preference attributes.

Following we present the SFM generated for the Dropbox example, which is shown in Figure §5.2 using a simplified stateful feature diagram:

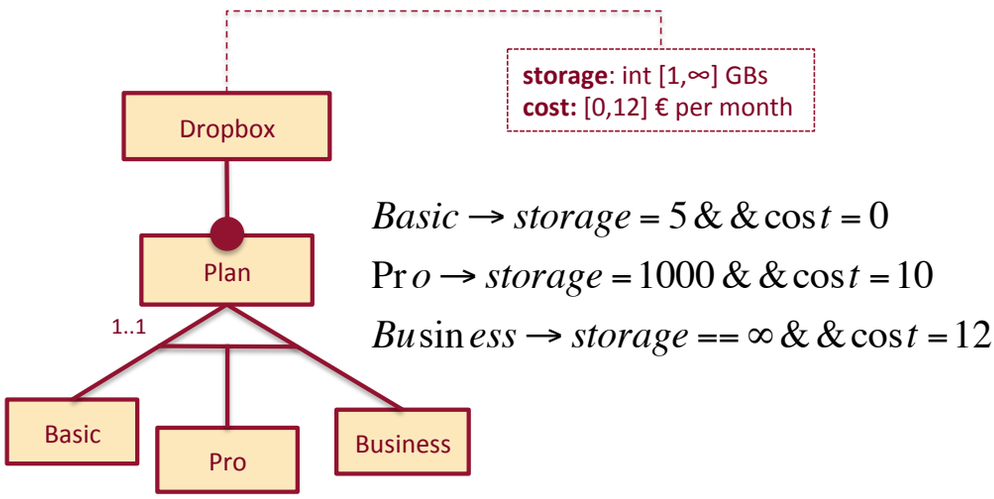


Figure 5.2: Dropbox SFM resulting from the mapping algorithm

$$
\begin{aligned}
E \times D = \{ \quad & F_{Root}, F_{plan}, F_{basic}, F_{pro}, F_{business} : \{sel, rem\}, \\
& F_{Root}.cost : \mathbb{R}, F_{Root}.storage : \mathbb{N}, \\
& F_{Root}.p, F_{basic}.p, F_{pro}.p, F_{business}.p : [0,1], \\
& F_{Root}.p_{cost}, F_{Root}.p_{storage} : [0,1]\} \\
R = \{ \quad & mandatory(F_{Root}, F_{plan}), alternative(F_{plan}, F_{basic}, F_{pro}, F_{business}) \\
& F_{Root}.cost \geq 0, F_{Root}.storage > 0, \\
& F_{basic} = 1 \Rightarrow F_{Root}.cost = 0 \wedge F_{Root}.storage = 5, \\
& F_{pro} = 1 \Rightarrow F_{Root}.cost = 5 \wedge F_{Root}.storage = 1000, \\
& F_{business} = 1 \Rightarrow F_{Root}..cost = 10 \wedge F_{Root}.storage = \infty, \\
& F_{Root}.p = (F_{basic}.p + F_{pro}.p + F_{business}.p + F_{Root}.p_{cost} + F_{Root}.p_{storage})/5\} \\
U = \{ \quad & F_{Root}.cost < 6, F_{Root}.storage > 20\}
\end{aligned}
$$

Table §5.2 summarises the rules to be applied in the mapping process.

5.2.3 Mapping HCSs to SFMs

An HCS is an aggregation of *Configurable Service (CS)* that are linked together by global terms and constraints. Mapping an HCS into an SFM implies at first mapping all the CSs within into separate SFMs. Specifically, a separate SFM is created for each instance or item of the CS, enabling a different set of user needs for each instance, according to the I set. Let us consider the Amazon example, with a configuration of two EC2 instances and one EBS instance as follows:

$map_{CS} : CS \times R^U \mapsto SFM$	
Common elements	
	Root feature $E = E \cup F_{Root} : \{sel\}$ **Preference attribute** $E = E \cup F_{Root}.p : [0,1]$
Configuration space	
Selectable terms $S = \{S_1,...,S_n\}$	**Abstract features** $F_S = \{F_{S,1},...,F_{S,n}\}, E = E \cup F_S$ $\forall F_{S,i} \in F_S \cdot R = R \cup mandatory(F_{Root}, F_{S,i})$
Selectable terms values $C = \{\{c_{1,1},...,c_{1,j}\},...,\{c_{n,1},...,c_{n,j}\}\}$	**Leaf features** $F_C = \{F_{S_1,1},...,F_{S_n,j}\}, E = E \cup F_C$ **Relationships** $\forall F_{S,i} \in F_S \cdot R = R \cup alternative(F_{S,i}, F_{S_i,1},...,F_{S_i,j})$ **Attributes** $\forall F_{S,i} \in F_S \cdot \{F_{S,i}.p : [0,1]\}, E = E \cup p_{S_i}$
Derived terms	
Terms and values $D = \{D_1,...,D_m\}$ $V = \{V_1,...,V_m\}$	**Attributes** $A = \{F_{Root}.att_{D_1} : V_1,...,F_{Root}.att_{D_m} : V_m\}, E = E \cup A$ $\forall D_i \in D \cdot \{F_{Root}.p_{D_i} : [0,1]\}, E = E \cup p_{D_i}$
Service constraints	
Service constraints R^S in $C \times V$	**Relationships** Constraints on F_S and/or A
Consumer needs	
Requirements R^U in $C \times V$	**User decisions** Constraints on F_S and/or A
Preferences	**User decisions**
$Favorites(S_i)$	$f = sel \Leftrightarrow f.p_{S_i} = 1 \wedge f = rem \Leftrightarrow f.p_{S_i} = 0$
$Dislikes(S_i)$	$f = rem \Leftrightarrow f.p_{S_i} = 1 \wedge f = sel \Leftrightarrow f.p_{S_i} = 0$
$Highest(D_i)$	$F_{root}.p_{D_i} = \frac{A_{ij} - A_{ij}^{min}}{A_{ij}^{max} - A_{ij}^{min}}$
$Lowest(D_i)$	$F_{root}.p_{D_i} = \frac{p_{max} - A_{ij}}{A_{ij}^{max} - A_{ij}^{min}}$
$Around(D_i,\text{v})$	$F_{root}.p_{D_i} = \frac{max(v - A^min_{ij}, A^max_{ij} - v) - \lvert A_{ij} - v \rvert}{max(v - A^min_{ij}, A^max_{ij} - v)}$
Preference structure	**Relationship**
$\langle C \times V, \gtrsim \rangle$	$F_{Root}.p = \frac{\sum_{F_i \in F} F_i.p_{S_i} + \sum_{D_i \in D} F_{Root}.p_{D_i}}{\lvert F \rvert + \lvert D \rvert}$

Table 5.2: Mapping configurable services into SFMs

$$\begin{aligned} CS &= \{ \quad CS_{ec2}, CS_{ebs} \} \\ I &= \{ \quad \{R^U_{ec2,1}, R^U_{ec2,2}\}, \{R^U_{ebs,1}\} \} \end{aligned}$$

In this case, each configurable service instance generates a SFM that joins the CS together with a set of user needs. The so-obtained SFMs are joined together with a unique SFM as follows:

$$SFM = map_{CS}(CS_{ec2}, R^U_{ec2,1}) + map_{CS}(CS_{ec2}, R^U_{ec2,2}) + map_{CS}(CS_{ebs}, R^U_{ebs,1})$$

Where $+ : SFM \times SFM \to SFM$ is a function that joins any two SFMs as follows:

$$(E_1, D_1, R_1, U_1) + (E_2, D_2, R_2, U_2) = (E_1 \cup E_2, D_1 \cup D_2, R_1 \cup R_2, U_1 \cup U_2)$$

The resulting SFM has three different roots, that must be joined by means of a unique root feature. For that sake, we create a F_{HCS} root feature that is linked to the root of each instance by means of a mandatory relationship. It results as follows:

$$\begin{aligned} E \times D &= \ldots \cup \{ \quad F_{HCS}, F^{ec2,1}_{Root}, F^{ec2,2}_{Root}, F^{ebs,1}_{Root} : \{sel, rem\}, \\ R &= \ldots \cup \{ \quad mandatory(F_{HCS}, F^{ec2,1}_{Root}), \\ &\qquad\quad mandatory(F_{HCS}, F^{ec2,2}_{Root}), \\ &\qquad\quad mandatory(F_{HCS}, F^{ebs,1}_{Root}) \} \end{aligned}$$

Let us consider an example where a term *subtotal* is defined as the sum of the cost of all the instances in I and *total* is a term that helps to define a discount policy, that applies a 10% discount for orders higher than 1,000$. This information is described in the HCS as follows:

$$\begin{aligned} T &= \{ \quad total, subtotal \} \\ V &= \{ \quad \mathbb{R}, \mathbb{R} \} \\ R^M &= \{ \quad subtotal = \textstyle\sum_{ij} cost_{ij}, \\ &\qquad\quad subtotal > 1{,}000 \Rightarrow total = 0.9 \cdot subtotal, \\ &\qquad\quad subtotal \leq 1{,}000 \Rightarrow total = subtotal \} \end{aligned}$$

The terms in the HCS are mapped into attributes in their corresponding domain. As in for CSs, a traceability table must be constructed in order to keep track on the elements in the SFM that corresponds to elements in the HCS. It would help in tracing the results of AASFM operations back into HCS terms. Table §5.3 shows an example

Dropbox Traceability Table	
Selectable Term	**Feature**
HCS	F_{HCS}
$R^{U}_{ec2,1} \in I_{ec2}$	$F^{ec2,1}_{Root}$
$R^{U}_{ec2,2} \in I_{ec2}$	$F^{ec2,2}_{Root}$
$R^{U}_{ecs,1} \in I_{ebs}$	$F^{ebs,1}_{Root}$
Derived Term	**Attribute**
$subtotal \in T$	$F_{HCS}.subtotal$
$total \in T$	$F_{HCS}.subtotal$

Table 5.3: Traceability table between HCS and SFM elements for the Dropbox example

of traceability table for our example. The following elements and relationships are created from the above information:

$$\begin{aligned} E \times D &= \ldots \cup \{ \quad F_{HCS}.total : \mathbb{R}, F_{HCS}.subtotal : \mathbb{R}\} \\ R &= \ldots \cup \{ \quad F_{HCS}.subtotal = F^{ec2,1}_{Root}.cost + F^{ec2,2}_{Root}.cost + F^{ebs,1}_{Root}.cost, \\ & \qquad F_{HCS}.subtotal > 1{,}000 \Rightarrow F_{HCS}.total = 0.9 \cdot F_{HCS}.subtotal, \\ & \qquad F_{HCS}.subtotal \leq 1{,}000 \Rightarrow F_{HCS}.total = F_{HCS}.subtotal\} \end{aligned}$$

Last, an overall preference variable is created and assigned to the F_{HCS} feature. Its value is the average of the p factor of each instance in the HCS:

$$\begin{aligned} E \times D &= \ldots \cup \{ \quad F_{HCS}.p : [0,1]\} \\ R &= \ldots \cup \{ \quad F_{HCS}.p = F^{ec2,1}_{Root}.p + F^{ec2,2}_{Root}.p + F^{ebs,1}_{Root}.p \end{aligned}$$

Table §5.4 summarises and generalises the rules to be applied in the mapping process.

5.3 Configurable Service Analysis operations

Mapping a configurable service into a SFM enables the use of analysis operations in the AASFM catalogue to extract relevant information from a configurable service. In this Section we present a catalogue of CS analysis operations, distinguishing between core and compound operations. A *core operation* cannot be defined in terms of any other operation. Each of the proposed core operations is solved relying on AASFM operations as shown in the following subsection.

5.3.1 Core operations

Configurations Listing

Finding all the configurations that can be made for a configurable service can also be defined in terms of an AASFM operation. Finding all the products in the SFM, we could obtain all the configurations in a CS. But the result of this operations is a set of products, each of which is an assignment of states for each element in the model (*sel* or *rem* for a feature, a value in the domain for an attribute, ...). So we need a $map^{-1} : \mathcal{P}(E) \rightarrow C \times V$ function that maps products in the SFM with configurations in the configurable service. The implementation of this function relies on the traceability table created across de mapping process. The all configurations operation can be defined as follows:

Operation 1 - Listing.
Let CS be a configurable service and R^U a set of user needs. Let $products : SFM \rightarrow D$ be an operation defined in the basic catalogue of the AASFM, which obtains all the products for a given SFM. The *listing* operation is defined as follows[1]

$$listing(CS, R^U) \rightleftharpoons products(map_{CS}(CS, R^U))$$

Configuration Ranking

Searching for the best configurations given a set of user needs is a very important operation. User requirements and preferences help a consumer to describe their needs in qualitative and quantitative terms. The proposed mapping provides a semantics for the preferences that enable the ranking of all the products in terms of its p attribute. The AASFM has an optimisation operation that ranks all the products according to a given criterion. In our case, the optimisation criterion is the maximisation of the p value, since $p = 1$ represents the top most preference. According to this, we can define the configuration ranking operation as follows:

Operation 2 - Configuration Ranking.
Let CS be a configurable service and R^U a set of user needs. Let $optimise : SFM \times \langle D, \gtrsim \rangle \rightarrow D$ be an operation defined in the basic catalogue of the AASFM, which ranks all the products for a given SFM according to a $\langle D, \gtrsim \rangle$ criterion. The configuration ranking

[1] Tracing the results from the AASFM back to terms in the CS domain is a recurring operation. For the sake of simplicity, we use the $\rightleftharpoons$ symbol to avoid any reference to mapping functions.

operation is defined as follows:

$$configRanking(CS, R^U) \rightleftharpoons optimise(map_{CS}(CS, R^U))$$

Inconsistent Configuration Explanation

When a configuration is detected to be invalid or no product at all is obtained when performing the listing or configuration ranking operations, we still need an explanation why is it not possible to find any valid configuration. The AASFM catalogue provides an operation to obtain explanations why a given set of user decisions is not valid, providing a set of minimal (and therefore most possible) explanations that could help the consumer in the correction of the configuration. An inconsistent configuration explanation can be defined in the following terms:

Operation 3 - Inconsistent configuration explanation.
Let CS be a configurable service and R^U a set of user needs. Let $whyNotConf : SFM \rightarrow U$ be an operation defined in the basic catalogue of the AASFM, which obtains the minimal explanations why a configuration is not valid. The wrong configuration explanation is defined as follows:

$$explainConfig(CS, R^U) \rightleftharpoons whyNotConf(map_{CS}(CS, R^U))$$

Inconsistent Service Explanation

In this last operation, the explanation is obtained in terms of the user needs that must be relaxed or removed to be able to find any solution to an invalid configuration. Sometimes the CS defines no configuration at all due to contradicting dependencies. These erroneous dependencies must be detected in order to repair the CS model. The AASFM catalogue also provides an operation to explain why an SFM is void due to contradicting relationships. Since relationships correspond to dependencies in the CS, this operation can be used to obtain the minimal explanations why a CS is invalid in terms of the conflicting dependencies. This operation can be defined in the following terms:

Operation 4 - Wrong service explanation.
Let CS be a configurable service. Let $whyNotRels : SFM \rightarrow R$ be an operation defined in the basic catalogue of the AASFM, which obtains the minimal explanations why an SFM is not valid. The wrong service explanation is defined as follows: $map_S^{-1} : R \rightarrow R^S$

$$explainService(CS) \rightleftharpoons whyNotRels(map_{CS}(CS, \emptyset))$$

5.3.2 Compound operations

The CS core operations are implemented relying on AASFM operations. However, there exist many other useful operations that can be defined on top of core operations. Next we present a non-exhaustive list with some examples of compound operations:

Operation 5 - Redundant dependency.
Let CS be a configurable service and $R_i^S \in R^S$ a service constraint defined in CS. Let $remove : CS \times R \rightarrow CS$ be a function that removes a constraint from the appropriate configurable service in an HCS. The redundant dependency operation can be defined in terms of the AASFM as follows.

$$redundant(CS, R_i^S) \equiv configurations(MS, \emptyset) = configurations(remove(CS, R_i^S), \emptyset)$$

Operation 6 - Dead value.
Let CS be a configurable service and $C_{i,j} \in C_i$ be a value for the S_i selectable term, both of them defined in CS. The term $C_{i,j}$ is dead if it is not possible to select it in a configuration, so this operation can be solved simulating a consumer that demands such value, being defined as follows:

$$dead(CS, C_{i,j}) \equiv \neg valid(CS, \{S_i = C_{i,j}\})$$

Operation 7 - False decision term.
Let CS be a configurable service and $C_{i,j} \in C_i$ be a value for the S_i selectable term, both of them defined in CS. The term $C_{i,j}$ is false if it must be chosen in every configuration in the decision space, so this operation can be solved simulating a consumer that demands any value other than the false one. This operation is therefore defined as follows:

$$falseDecision(CS, C_{i,j}) \equiv \neg valid(CS, \{S_i \neq C_{i,j}\})$$

Operation 8 - Inconsistent service.
Let CS be a configurable service. This CS is not configurable if it defines no configuration at all due to contradictory constraints, which can be detected as follows:

$$inconsistent(CS) \equiv count(CS, \emptyset) = 0$$

Operation 9 - False configurable service.
A configurable service is false configurable if it only defines one possible configuration. In other words, the cardinal of the configuration space is exactly one. This operation can be defined as follows:

$$falseConfigurable(CS) \equiv count(CS, \emptyset) = 1$$

Operation 10 - Configurations counting.
Let CS be a configurable service. This operation returns the total number of configurations existing in the decision space of the CS.

$$count(CS) \equiv |configurations(CS, \emptyset)|$$

Operation 11 - Valid requirements.
Let CS be a configurable service and R^U a set of user requirements to be validated. This operation checks if the given user requirements are valid for the CS.

$$validReqs(CS, R^U) \equiv |configurations(CS, R^U)| > 0$$

Operation 12 - Best configuration.

$$best(CS, R^U) \equiv configRanking((CS, R^U), max\{F_{Root}.p\})$$

Operation 13 - Explain dead value.
Let CS be a configurable service with a dead value $C_{i,k}$ for the term D_i. This operation returns the minimal set of explanations which makes $C_{i,k}$ a dead value.

$$explainDeadValue(CS, C_{i,k}) \equiv explainConfig(CS, S_i = C_{i,k})$$

5.4 HCS Analysis operations

5.4.1 Core operations

Operation 1 - Valid configuration.
We can define the valid configuration operation relying on the $valid : SFM \rightarrow \{true, false\}$ operation defined in the basic catalogue of the AASFM as follows:

$$validConfiguration(HCS) \equiv valid(map_{HCS}(HCS))$$

Operation 2 - All configurations.
We can define the valid configuration operation relying on the $products : SFM \rightarrow D$ operation defined in the basic catalogue of the AASFM, which obtains all the products for a given SFM, as follows:

$$configurations(HCS) \rightleftharpoons products(map_{HCS}(HCS))$$

Operation 3 - Best configurations.
... $optimise : SFM \times \langle D, \gtrsim \rangle \rightarrow D$ operation defined in the basic catalogue of the AASFM, which ranks all the products for a given SFM according to a $\langle D, \gtrsim \rangle$ criterion, as follows:

$$best(HCS) \rightleftharpoons optimise(map_{HCS}(HCS), max\{F_{HCS}.p\})$$

Operation 4 - Wrong configuration explanation.
... $whyNotConf : SFM \rightarrow U$ operation defined in the basic catalogue of the AASFM, which obtains the minimal explanations why a configuration is not valid, as follows:

$$explainConfig(HCS) \rightleftharpoons whyNotConf(map_{HCS}(HCS))$$

5.4.2 Compound operations

Suppose we have a $reset : HCS \rightarrow HCS$ function that empties all the elements in I, removing any configuration.

Operation 5 - Valid HCS.

$$validHCS(HCS) \equiv validConfiguration(reset(HCS))$$

5.5 Summary

In this chapter, we have presented our approach to automate the analysis of configurable and highly-configurable services. For this purpose, we have provided HCSs with formal semantics through a mapping to SFMs, which enable the use of the AASFM as a basis for the automated analysis of HCSs. A set of core operations for the HCSs–

operations that cannot be expressed in other way than in terms of the AASFM– have been defined on top of the core operations of the AASFM. And later, a set of compound analysis operations have been built on top of the core HCS operations. In this way, we assist the HCS validity criteria described in Section §4.4, the decision making of users on HCSs. The automated analysis also provides explaining operations for the errors found in both tasks and algebraic operations for the verification and comparison of HCSs.

$map_{HCS} : HCS \mapsto SFM$	
Common elements	
	Root feature $E = E \cup F_{HCS} : \{sel\}$ **Preference attribute** $E = E \cup F_{HCS}.p : [0,1]$
Configurable services	
Items $CS = \{CS_1, ..., C_k\}$ $I = \{I_1, ...I_k\}$	**SFM** $\forall I_i \in I, \forall R_j^U \in I_i \cdot map_{CS}(CS_i, R_j^U)$
Items $I = \{I_1, ...I_k\}$	**Constraints** $\forall I_i \in I, \forall R_j^U \in I_i \cdot R = R \cup \{mandatory(F_{HCS}, F_{Root}^{ij})\}$
HCS Terms	
Terms and values $T = \{T_1, ..., T_m\}$ $V = \{V_1, ..., V_m\}$	**Attributes** $A = \{F_{HCS}.att_{D_1} : V_1, ..., F_{HCS}.att_{D_m} : V_m\}, E = E \cup A$ $\forall D_i \in D \cdot \{F_{HCS}.p_{D_i} : [0,1]\}, E = E \cup p_{D_i}$
HCS Constraints	
Dependencies R^M	**Relationships** Constraints on A
Consumer needs	
Preference structure $\langle C \times V, \gtrsim \rangle$	**Relationship** $F_{HCS}.p = \frac{\sum_{I_{i,j} \in I} F_{Root}^{I_{i,j}}.p}{\lvert I_{i,j} \rvert}$

Table 5.4: Mapping HCSs into SFMs

Part IV

Validation

Automated Configuration Support for Infrastructure Migration to the Cloud

The measure of intelligence is the ability to change.

Albert Einstein (1879 - 1955),
Theoretical physicist

In this chapter, we present the content of the paper accepted for publication in the Future Generation Computer Systems of Elsevier. Section §6.1 introduces the paper, while Section §6.2 briefly describes the state of the art in variability modelling and analysis. Section §6.3 states the problem we tackle in this work. Section §6.4 describes our modelling methodology for the configuration space of an IaaS, while Section §6.5 presents a modelling case study with Amazon EC2. Section §6.6 presents our analysis approach for the search of the optimal configuration, and explains the details of the analysis operations. We present an implementation of our approach in Section §6.7, and we evaluate it in Section §6.8. Related work is described in Section §6.9. Finally, Section §6.10 discusses our proposal and proposes future directions in our research.

6.1 INTRODUCTION

The clear benefits of cloud-based infrastructures are increasing the number of companies that are migrating their private and expensive data centers to the cloud. An IaaS enables the dynamic provisioning of computational & data resources (often on-demand), reducing costs (for short term workloads), speeding up the start-up process for many companies, and decreasing resource and power consumption (among other benefits).

Deciding the most suitable provider is often challenging, as each provides a number of possible configurations. As an example of the dimension of this problem, there are over 100 public cloud providers [36], and just for EC2 [5], the *Amazon Web Services (AWS)* computing service, we have identified 16,991 different configurations[1]. As each user/company that plans to use a cloud computing infrastructure is likely to have their own specific requirements, it is necessary to identify the most relevant provider and subsequently the most suitable configuration. Identifying such configuration within a large potential search space is a tedious and error-prone task that requires for an automated support.

In recent times, software tools and research contributions have emerged to support this decision process, but we have found these to have limitations, providing either incomplete configuration spaces or/and imprecise results. On the commercial side, providers such as Amazon or Rackspace provide tools that suggest specific configurations for migrating an on-premise infrastructure [3, 130]. However, such tools ignore some configuration options, forcing for example in the case of Amazon to choose Linux as the only operating system. Other companies, like CloudScreener [38], provide their own comparators to decide which provider and configuration best fit user needs. Nonetheless, such tools lack information about the configuration space they work with, and some tests have revealed false positives in their optimal results. Recent academic work [58, 151] also suffers from similar concerns, as they generally only consider a small subset of the available configurations of services like EC2 or Azure virtual machines. In order to overcome these limitations, any approach must ensure for each provider to model its complete configuration space.

In this work, we assist the customer in determining the most suitable configuration of an IaaS. For that purpose, we present the case study of AWS EC2. As outlined previously by us [69], AWS is one of the most variable and complex providers in terms of

[1] The basis for this number is explained further in this paper.

configuration options and pricing. Indeed, understanding EC2 configuration space can be challenging, as it is scattered across several pages, tables and paragraphs. We believe that modelling a complex provider eases the task of modelling simpler providers. Additionally, AWS is one of the most widely used IaaS providers, being present in all the current configuration tools. For focusing on one provider, enables us to check their precision and compare to our approach. Among all the different services AWS offers, we focus on EC2 and EBS– additional disk for computing instances, which are considered as core infrastructure services.

We interpret an IaaS as a *variability-intensive* system, so we can rely on variability modelling and analysis techniques to support the configuration process. In particular, we propose the modelling of IaaS– and EC2 in concrete – as FMs, a kind of model widely used for variability-intensive systems. In this way, first we represent the configuration space in a complete, structured and compact manner, and second we provide the user with a model to ease the configuration process. This modelling enables the use of the so-named AAFM, a set of analysis operations that extracts information from the models, which we subsequently use to assist decision-making. We use some of them to verify the validity and completeness of the FM with respect to the service configuration space, and to determine which configuration is the most suitable for any given requirements. We interpret the most suitable configuration as the one that meets customer's requirements and optimises the cost. Our approach presents two main benefits: first, we consider the complete configuration space, so that the real optimal solution is obtained for given customer requirements; second, assisting the configuration process for such a highly-configurable service like AWS EC2 enables the same approach to be used for other providers, such as Azure or Rackspace.

For evaluation purposes, we *(i)* verify our proposed model, *(ii)* compare our approach to existing commercial applications in term of expressiveness and accuracy, and *(iii)* check and improve the performance of our approach. To verify our model, we describe the EC2 FM using a plain-text language, extract the list of configurations within our model, and check that it matches exactly the available configurations of EC2[2]. The validation of the analysis is performed by means of two different implementations: FaMa Framework – a well-known tool for the AAFM– and a reasoner based on the IBM CPLEX solver. We compare the performance of both approaches, where the later implementation shows improved and negligible execution times when calculating the most suitable configuration. We also compare the obtained results with the output of

[2]We exclude spot instances and micro instances since they are not intended to be persistent, and EBS optimised instances because we do not consider IOPS provisioning for EBS.

CloudScreener, which can be improved by the use of our approach.

This paper extends our previous work [69] in several ways. In particular, we provide i) an explicit description of the configuration problem, ii) a modelling methodology to describe the configuration space of an IaaS as a FM, iii) a verification of the configuration space represented by the FM by means of analysis operations and iv) an evaluation of the expressiveness, accuracy and performance of our approach.

The rest of the paper is structured as follows: Section §6.2 briefly describes the state of the art in variability modelling and analysis. Section §6.3 states the problem we tackle in this work. Section §6.4 describes our modelling methodology for the configuration space of an IaaS, while Section §6.5 presents a modelling case study with Amazon EC2. Section §6.6 presents our analysis approach for the search of the optimal configuration, and explains the details of the analysis operations. We present an implementation of our approach in Section §6.7, and we evaluate it in Section §6.8. Related work is described in Section §6.9. Finally, Section §6.10 discusses our proposal and proposes future directions in our research.

6.2 Feature Models

Feature Models (FMs) [88] are used to represent all the possible products that can be built in variability-intensive systems such as SPLs. FMs are tree-like data structures where each node represents a product feature. Figure §6.1 shows a FM that represents general features of a fictional IaaS provider. Features are bound by means of hierarchical (mandatory, optional and set) and cross-tree relationships. These relationships define how features can be combined in a product, defining the configuration space of the system. In a FM, a feature does not necessarily represent a specific functionality but can be used as abstract features [153] which represent domain decisions such as `Linux based` feature (Figure §6.1). `IaaS` is the root feature that represents the overall functionality of the system. It has two children, an optional feature (white circle) named `Storage`, and a mandatory feature (black circle) named 0S. Both features present set relationships whose cardinality indicates the number of child features that can be chosen at the same time.

FMs can also have attributes that represent non-functional properties, leading to attributed FMs. These attributes are linked to a specific feature. In the FM sample, the size attribute is linked to the `Storage` feature, and monthly cost attribute is linked

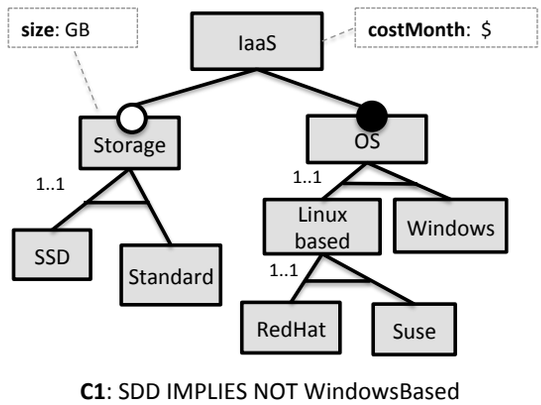


Figure 6.1: Example of a FM

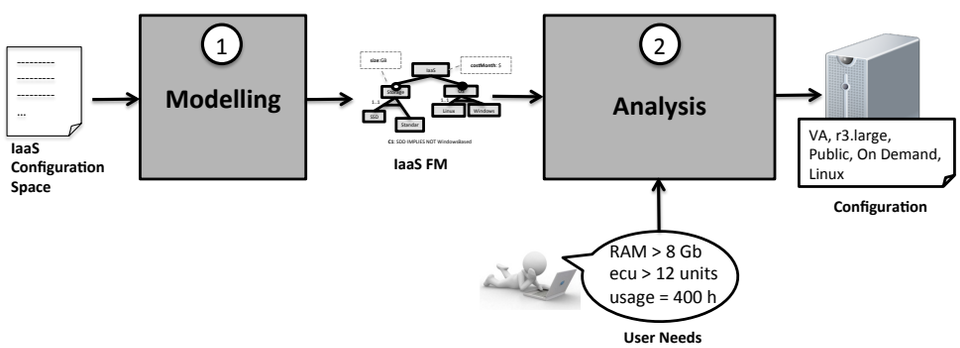


Figure 6.2: IaaS Configuration Space Description and Analysis as a FM.

to the root feature. Optionally, it is possible to define constraints that describe the relationships among attributes.

FMs contain valuable information about all the possible configurations and their properties. From a FM we can deduce a number of possible outcomes, such as the total list of possible products, the set of common features among products, the set of products that meet a given criterion and the product with a minimum cost. Analysing the FM manually is a tedious and error-prone task. Many researchers have focused on the AAFM [18, 19], that currently offers over 30 different analysis operations, each of them solved by means of different declarative approaches such as constraint programming, SAT problems or binary decision diagrams.

6.3 PROBLEM

Migrating existing applications to the cloud has received significant interest recently. Cloud benefits, like cost savings, scalability and on-demand features, make large companies and SME want to embrace the cloud. However, this process requires facing a number of issues, such as the need to carry out feasibility studies, provider selection or code/ application modifications. Recently, a *Cloud Reference Migration Model* has been proposed [86] to address such issues. This reference model encompasses three main phases: planning, execution and evaluation.

Selecting the most suitable provider and its configuration are relevant decisions in the migration planning process. Each provider offers several configuration options, each with up to dozens configuration values, which may be slightly or very different from one vendor to another. Moreover, the information and constraints about the configuration space are often poorly organised. For instance, the configuration options and values of EC2 are scattered among three different pages, making it difficult to manually search for suitable configurations. Rackspace presents the configuration space of its servers also in a similar way. The size, interconnections and organisation of the configuration space makes configuration a tedious and error-prone process requiring an automated solution.

In this work, we identify and tackle two challenges, as depicted in Figure §6.2

1. *Model the configuration space of a cloud provider in a structured and compact way.* Configuration information is commonly described in an unstructured way using natural language, and therefore subject to ambiguity. It is necessary to model and structure it to enable ease of use and understanding of the services by an end-user, and to enable automated support. Most of the academic works that have approached migration issues [24, 58, 128, 150, 151, 163, 165] deal with reduced subsets of the providers' configuration spaces. However, when we plan for cloud adoption (or migration), we have to address the whole configuration space to find the most suitable configuration.

2. *Assist the search of the most suitable configuration.* Even with a well-structured configuration space, most of the providers offer thousands of different configurations. Assisting the search for the most suitable configuration would save time and effort for the users, and could ensure that the selected configuration really fits customer needs. In this paper, we define the most suitable configuration as

Servers and characteristics							
Purpose	**Instances**	**Cores**	**RAM**	**Disk**	**Location**	**OS**	**Hours / Months**
Production	2	2	4	2 TB	Europe	Linux-based	730 / 12
Pre-production	1	2	4	250 GB	-	Linux-based	160 / 12
Perf. testing	1	8	4	1 TB	-	RedHat	40 / 12

Table 6.1: Migration case study

the one that fulfils the customer requirements and minimises the total cost.

6.3.1 Scenario

We present a migration scenario of a SME, which has multiple application servers and a server for pre-production tasks. They plan to migrate to the the AWS cloud, and they look for servers with the characteristics shown in Table §6.1. In addition to the the two application and one pre-production servers, they are currently involved in new developments that require performance testing. Therefore, they also need a large compute-intensive instance for testing. The application instances must be running 24/7, while the pre-production instance is needed 40 hours per week. The performance testing instance will be used around 40 hours per month.

We have chosen AWS and EC2 as the provider and service under study for several reasons. First, AWS is one of the most widely used providers of infrastructure services. Second, the number of configuration options and values of EC2 leads to thousands of different configurations that necessitates automated assistance. Finally, the complex price policy turns the modelling into an even more challenging task.

6.4 MODELLING

In this section, we describe our approach to model an IaaS configuration space as a FM. First, we present a brief taxonomy of the typical IaaS configuration options offered by commercial providers. We then propose a modelling methodology to describe such configuration options as a FM.

6.4.1 IaaS Configuration Options

IaaS commercial providers aim to address the needs of different kinds of customers. From those looking for on-demand virtual machines to big companies such as Netflix requiring massive computation and storage, IaaS providers offer them multiple configuration options to meet their requirements. For this taxonomy of the IaaS configuration space, we have studied four main providers: Google [75], Amazon [5], Microsoft [112] and Rackspace [131]. Most of the providers enable configuration options based on the following factors:

- *Instance type*. Each instance type determines the basic characteristics of the computing instance – RAM and number of cores. Depending on the provider, a default storage, linked to the instance, is also offered. The instance type range is usually broad, from small instances with a single core to large clusters.
- *Operating system*. Most of the providers offer different flavours of Windows and Linux OS.
- *Storage*. Although some providers offer a default storage depending on the instance type, all of them allow to hire additional storage linked to the instance. This storage is usually limited to a number of GBs per instance, and can be based on Solid State Disks (SSDs).
- *Geographic location*. The infrastructure services offered by IaaS providers are delivered through different datacenters, spread across multiple geographic locations around the world.
- *Purchasing mode*. While all the providers offer their services on-demand, usually billing the use per hour, most of them also provide additional commitment plans or purchasing modes to hold their customers while they get significant savings.

6.4.2 Modelling Methodology

In order to describe the aforementioned configuration options as a FM, we provide a description about the method we follow. Lee *et al.* [96] propose some guidelines to identify and organise features that we extend to support attribute modelling as follows:

1. *Tree definition*. The tree structure of the FM, i.e. the features and their relationships, is defined at first.

Mapping		
EC2	**Example**	**FM**
Config point	Location	Abstract feature
Config value	US Virginia	Leaf feature
Usage data	Months	Attribute
Service att.	Cost hour	Attribute

Table 6.2: Iaas mapping to EFM

(a) *Features definition*. A feature [88] is a tree node which can be selected or removed. Traditionally, two types of features have been considered: leaf features to express the configuration values that define the configuration space, and abstract features to organise semantically related features. We interpret all the values that make any two IaaS configurations to be different as leaf features in the FM. We also identify a number of abstract features, that we will define as the configuration options. These features may also be grouped, such OS, geographical location, or instance type – see Table §6.2.

(b) *Relationships definition*. Once we have defined the leaf features and the configuration options and its possible values from the provider documentation, we create the tree like structure by means of relationships. The root feature is placed at the top representing the whole configuration space of the IaaS. At a second level, all configuration options are placed as children of the root feature, having either a mandatory or an optional relationship depending on the optional character of the feature. At a third level, all the leaf features are placed as child of the corresponding configuration option. Since only one leaf feature can be selected at a time for each configuration option, they are grouped by means of a set relationships with a 1..1 cardinality. Once all the leaf features are placed in the model, we introduce some intermediate abstract features to ease configuration and to represent higher-level decisions.

2. *Attributes definition*. At this point, there exist elements that cannot be modelled as features, because their domain is not boolean and/or because their values is the result of a calculation from other attributes or features. For example, an instance cost per hour is clearly a non-boolean value that depends on the cost of the selected features. Such elements are modelled as attributes – with units and domain – and are linked to their related feature. We also identify two kinds of attributes, as shown in Table §6.2:

- *Usage data:* information about the usage which is provided by the customer, e.g. instance hours per month.
- *Service derived attributes:* attributes whose values are obtained from other decisions, like the RAM size or number of cores of an instance, which depend on the instance type.

3. *Constraints definition*. Finally, we define constrains on the features and attributes of the FM to adequately represent the IaaS configuration space. These constraints encompass the instance cost hour, availability, characteristics and additional dependencies. For most of the cases, the only source to obtain constraints associated with the configuration space is the cloud provider's website. While the extraction of some constraints is trivial, others have to be inferred from the description of the service in natural language. An example of the former is the pricing of the instances, usually structured in tables, such as in Amazon, Rackspace or Microsoft Azure. For instance, consider the following constraint:

$$\texttt{(M3.large AND OneYear AND Sydney AND RedHat AND Public) IMPLIES (upfrontCost==249 AND costHour==0.235)}$$

In order to automate the extraction of this kind of constraints, we can use web scraping techniques. It is important to note that the configuration space of commercial providers evolves often, so manual adjustments in the scraper or advanced scraping techniques may be required in such cases. However, the extraction of other constraints, such the total cost per year of the infrastructure

$$\texttt{Total Cost = Cost Hour} \times \texttt{Hours Per Month} \times \texttt{Number Of Months + Upfront Cost + EBS cost}$$

cannot be automated, and must be formulated manually.

6.5 Modelling Case Study: Amazon EC2

In this section, we exemplify our modelling approach by means of a case study: the description of Amazon EC2 as a FM. First, we briefly describe the main aspects and configuration options of EC2. Second, we present the EC2 FM, result of the methodology described in the previous section. Finally, we suggest some guidelines for the configuration of an EC2 instance using the EC2 FM.

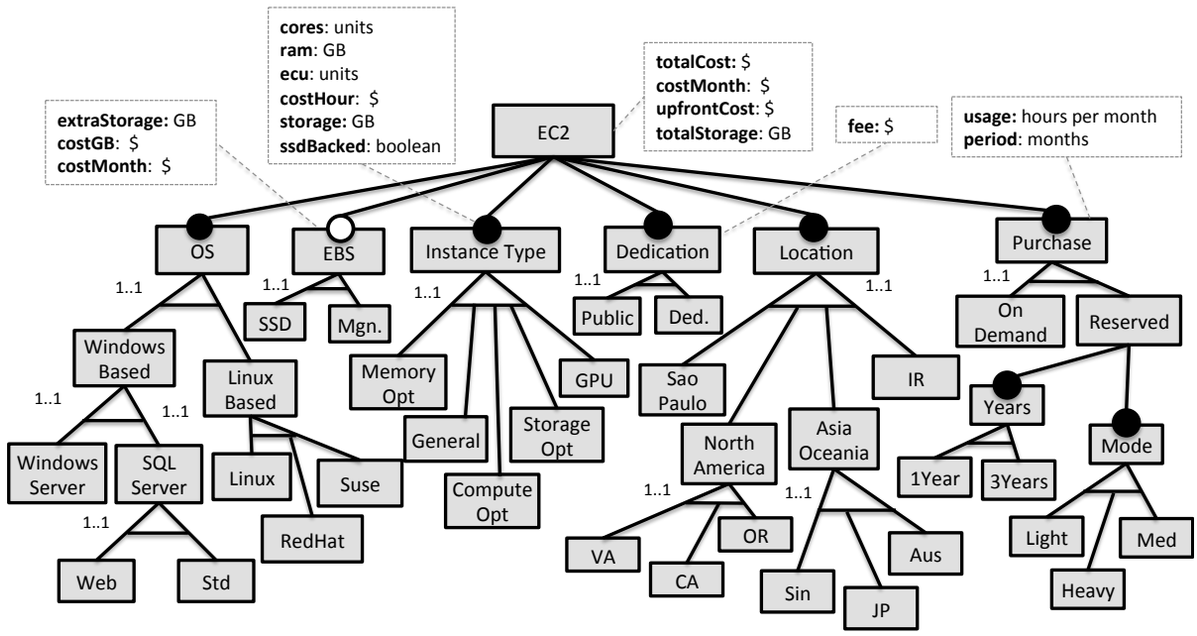


Figure 6.3: Feature Model of EC2 and EBS

6.5.1 AWS Elastic Compute Cloud

AWS provides services for computation, storage, databases, clusters or content delivery among others. Due to the wide range of services, several PaaS and SaaS providers like Heroku or Netflix run over AWS. As defined by Amazon, EC2 "provides resizable compute capacity in the cloud" [5] on pay-per-use basis. In this paper, we work with the AWS snapshot for EC2 and EBS of 12th June 2014. On this date, Amazon provides five main configuration options for EC2, which match the general configuration options of IaaS providers depicted in the previous section:

- *Instance type*. EC2 offers 32 different instance types, grouped under different categories, depending on the purpose: General Purpose, Compute Optimized, GPU Instances, Memory Optimized and Storage Optimized, distinguishing also between current and previous generations. Each instance type has a specific RAM size, a number of cores and disk storage. In this point, we ignore micro instances, since they are intended for short CPU burst purposes. Consequently, they lack disk storage, and their performance is highly variable and unpredictable.

- *Operating system*. Three different Linux distributions (Amazon Linux, Suse and Red Hat) and a Windows version (Windows Server) with an optional SQL Server in different flavours (Express, Web and Standard) are available.

- *Storage*. AWS provides a fixed disk size for EC2 instances, depending on each instance type. If the default storage needs to be extended, the EBS service provides additional storage. In this sense, we must say that we do not consider provisioned IOPS (input/output operations per second) EBS, and neither EBS optimised instances, that "*enable EC2 instances to fully use the provisioned on an EBS volume*".

- *Geographic location*. Amazon offers 8 geographic locations, distributed among different areas of North America, Europe, Asia and South America.

- *Purchasing mode*. EC2 instances may be purchased on demand or in a reservation way. While users pay per use in both modes, cost hour is lower for reserved instances in exchange for an upfront payment. In total, there are seven different purchasing modes. We exclude here spot instances, because Amazon determines their price and availability dynamically based on supply and demand. Indeed, AWS recommend the use of spot instances only for time-flexible and fault-tolerant tasks.

Additionally, an EC2 instance may run on a dedicated machine, guaranteeing an additional isolation in exchange of an additional cost.

6.5.2 EC2 Feature Model

The resulting EC2 FM is defined in a plain-text language and presents 81 features, 17 attributes and more than 20 000 constraints [3]. It is based on the description of EC2 of the day 12 June 2014 [5]. The huge amount of constraints needed to represent pricing and instances availability makes necessary to automate their extraction by means of a web scraper, whose details are explained in Section §6.7. Due to the difficulty of representing the complete model using feature diagrams, only an excerpt is shown in Figure §6.3.

The root feature, *EC2 Instance*, groups the configuration options and defines 4 attributes: *totalCost*, *costMonth*, *upfrontCost* and *totalStorage*. There are 6 configuration options where only one leaf feature can be selected at the same time: *(i) OS* is composed by Linux and Windows variants. We include some abstract features and SQL Server options for more flexibility in the decisions. Since SQL Sever Express is included by default in any Windows variant, we exclude it from the FM. *(ii)* The additional *Block*

[3] Available at https://dl.dropboxusercontent.com/u/1019151/EC2FM.pdf

Storage is defined as optional, and can be SSD or magnetic based. It has three attributes: *extraStorage* for the size, *costGB* for the GB cost per month, and *costMonth*, which value is calculated as $extraStorage \times costGB$. *(iii) Instance* contains general purpose, high mem, high CPU, high IO and GPU features. Due to spatial constraints the specific instances types are shown separately in Table §6.3. We define 6 attributes for this configuration option: *cores* for the number of cores of the instance, *ram* for the memory, *ecu* for the EC2 Compute Units, *costHour*, *defaultStorage* for the default storage of the instance, and *ssdBacked* to show if the instance storage is SSD backed. *(iv) Dedication* determines if the EC2 instance is based on shared or dedicated hardware resources. This kind of isolation implies additional cost, so we have defined an attribute, named *fee*, to represent it. *(v) Location* groups all the available locations for EC2 instances by continent and state/country. Finally, *(vi) Purchase* represents the purchasing options, and also defines use hours and number of months, by means of *usage* and *period* attributes.

Instance types	
Category	**Specific types**
General purpose	M3.medium, M3.larg. M3.xlarge, M3.2xlarge, M1.small, M1.medium, M1.large, M1.xlarge
Compute opt	C3.large, C3.xlarge, C3.2xlarge, C3.4xlarge, C3.8xlarge, C1.medium, C1.xlarge, CC2.8xlarge
Memory Opt	M2.xlarge, M2.2xlarge, M2.4xlarge, CR1.8xlarge, R3.large, R3.xlarge, R3.2xlarge, R3.4xlarge, R3.8xlarge
Storage Opt	I2.xlarge, I2.2xlarge, I2.4xlarge, I2.8xlarge, HS1.8xlarge, HI1.4xlarge
GPU	G2.2xlarge, CG1.4xlarge

Table 6.3: EC2 instance categories and specific types

6.5.3 Customer Requirements on EC2 FM

Representing the EC2 configuration space as a FM has several benefits. Our model represents the EC2 configuration space in a compact and well-structured way, so it is easy for the customer to see the big picture. Abstract features let users make high-

Requirements								
Instance	**Location**	**OS**	**Dedic.**	**Cores**	**RAM**	**Storage**	**Usage**	**Period**
Production	IR	LinuxBased	Public	2	4	2,000	730	12
Pre-production	-	LinuxBased	Public	2	4	250	160	12
Testing	-	RedHat	Public	8	4	1,000	40	12

Table 6.4: Customer requirements for the case study

level decisions, and attributes like RAM, storage and period add the ability to make decisions about user terms besides EC2 configuration terms. Moreover, FMs enable the use of the AAFM analysis operations to assist the decision-making.

We propose to represent customers' infrastructure requirements relying on the EC2 FM. Customers can make decisions on the different configurations points of our model. They can decide whether to select a feature or not, and specify preferences about attributes. At least, customers should make the next mandatory decisions:

- Assign values to attributes *period* and *usage*.

Optionally, we recommend costumers to make decisions on the following elements:

- Select a child feature of *OS*, *Dedication* and *Location*.
- Remove or select a child feature of *EBS*.
- Assign values to *ram*, *cores*/*ecu* (one of them) and *ssdBacked*.

We take the case study of Section §6.3.1 and Table §6.1 as an example. Table §6.4 shows the decision making we propose for each server. As we can see, we select abstract feature for some cases, which give us higher-level decisions, and specific features in others. For attributes, the value is constrained with a greater or equal relational constraint ($\geq$), since it is possible that EC2 cannot match exactly the value.

6.6 FM Analysis

Modelling the configuration space of an IaaS as a FM enables the extraction of information by means of AAFM operations. Benavides *et al.* [19] define the AAFM as "*the*

process of extracting information from FMs using automated mechanisms". The AAFM provides a catalogue of analysis operations, where each operation retrieves different kinds of data from the model. An analysis operation can be interpreted as a black-box procedure that receives a FM and any operation-specific parameter as inputs and retrieves a different kind of output data depending on the operation. Examples of AAFM operations are products listing (that enumerates all the products described by a FM), valid configuration (that checks whether it is possible to find at least one product given user requirements), or error checking (that searches for different kinds of semantic errors).

The main goal of this work is to automate the search of the most suitable configuration of a given IaaS provider – AWS EC2 in our case – that meets a set of user requirements. Prior to configuration search, we need verify that the FM correctly represents the configuration space of service. For that purpose, we identify two tasks to perform that can be solved relying on AAFM capabilities: (1) error checking, to ensure the absence of semantic errors [157] in the FM and (2) product listing, to obtain a list of all the possible configurations. In this section we detail the use of AAFM operations in order to accomplish these tasks.

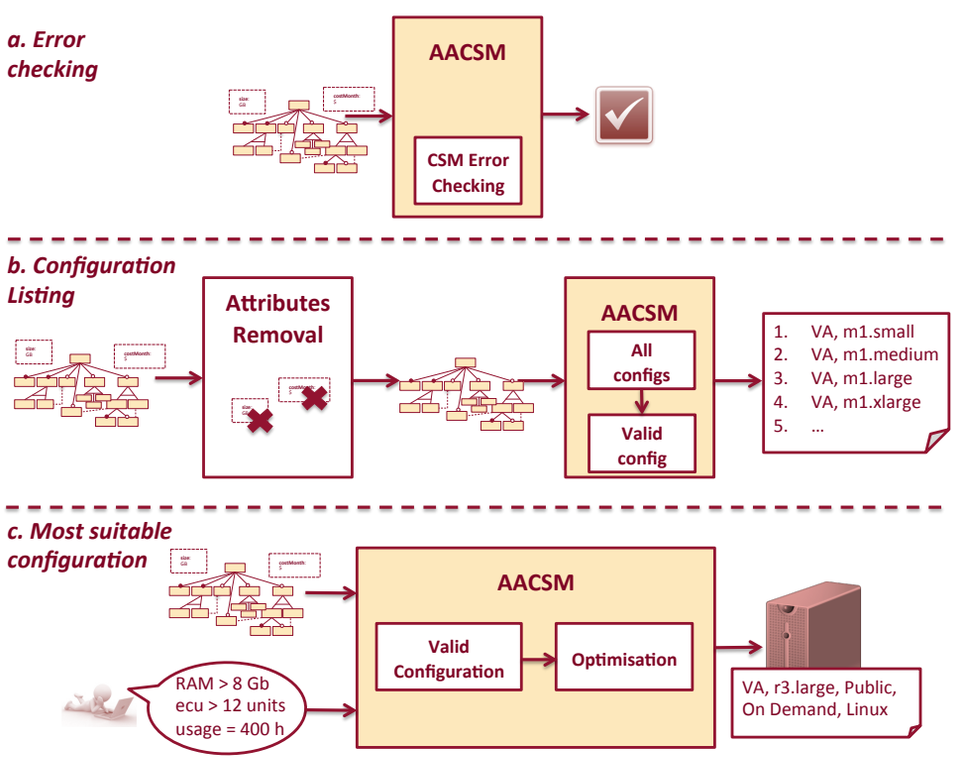


Figure 6.4: AAFM operations support for EC2 configuration

6.6.1 Error Checking

As Trinidad et al. propose [157, 174], it is a good practice to check that a FM is free of errors prior to perform any further analysis operation. When cross-tree constraints and attributes are used, contradictory information can be introduced in a FM provoking undesirable effects and making the FM not to represent the configuration space correctly. The AAFM provides the *Error Checking* operation (Figure §6.4.a) that determines if a FM has any of four kinds of errors:

1. Void FM: a FM that describes no product at all due to contradictory relationships or constraints.
2. Dead features: a feature that cannot be selected and therefore appears in no product.
3. False-optional features: a feature that despite of being modelled as optional, must be selected as far as its parent feature is selected being a *de facto* mandatory feature.
4. Wrong cardinals: any number of child features in a set relationship that cannot be selected at the same time.

Just in case any error were found, the AAFM provide an error explanation operation that proposes different reasons why the errors appear in a model. These explanations can be used to identify the source of the errors and repair them manually.

6.6.2 Configurations Listing

Despite error checking is a good practice that increases the confidence in the correctness of a FM, it does not guarantee that the FM represents all and just all the products in a configuration space. We need to list all the available configurations in the IaaS FM and check each of them against the retrieved information of the IaaS provider. We explain how we check them or the particular case of EC2 in Section §6.7.

Throughout this paper, we have been referring to the enormous size of the EC2 configuration space. In particular, we have referred to the potential number of configurations (22,848 for EC2), i.e. all the possible combinations of the different values of its configuration points, and to the available configurations (16,991). The reason is that not all the potential configurations are really available. For example, some locations

and purchasing modes exclude the availability of several instance types. In order to obtain the list of available configurations, we use the *All Products* operation [19]. This operation, as its name denotes, returns the list of products that a FM represents. But first we still need to match the concepts of IaaS configuration and FM product.

We consider a *configuration* as the set of required decisions to make in order to run a cloud service instance. In the case of an IaaS computing instance, such decisions match the configuration points previously defined: OS, instance type, dedication, location and purchasing mode. Additionally, for EC2 we have to decide also on the dedication. In a FM, a product is a set of selected features that satisfies all the relationships and constraints [19]. It is necessary to remark that this definition matches for basic FMs that contain no attributes. This makes necessary to review the product concept for attributed FMs.

In an attributed FM, users can not only make any decision about features, but also about attribute values. A product is defined when a decision is made about all the features and yet all the relationships and constraints are satisfied. Due to the reasoning techniques used to perform the AAFM, it is not possible to list the products defined by an attributed FM with precision. An approximated way to perform this operation consists of removing attributes and their constraints from the model to obtain a basic FM just with features and relationships. Using this basic FM, we can perform the All Products operation to get the list of the available products of an attributed FM.

In an attributed FM, attribute constraints could discard certain combinations of features because of the values of their linked attributes. When attributes and constraints are removed from the FM, these combinations are available again, so we cannot expect that the set of basic products coincides with the set of attributed products. In order to check if a basic product is also a valid attributed product, the *Valid Configuration* operation can be used to check if it satisfies all the constraints in the attributed FM. Repeating this operation for each basic product in the list, we obtain the list of attributed products that satisfies all the relationships and constraints in an IaaS FM.

Figure §6.4.b depicts the process to obtain the list of IaaS configurations. First, the IaaS FM is transformed into a basic FM, removing all the attributes and their related constraints. In the specific case of EC2, EBS feature is also removed, since in this point we are only interested in EC2 configurations. Afterwards, we invoke All Products operation, which returns the list of available basic products, and for each of them, we check if it is valid for the attributed model. In this way, we obtain the number of total EC2 configuration: 16,991.

6.6.3 Most Suitable Configuration

Once the FM is checked to define the IaaS configuration space correctly, we can perform any analysis operation with confidence. The main goal of this work is assisting the users in the search of the most suitable configuration. Previously, we have defined the most suitable configuration as the one that satisfies customer requirements while the cost is minimised. This problem can be interpreted as an optimisation problem, where user requirements are hard constraints and the optimisation criterion is minimising the cost.

Prior to solving any optimisation problem, we should check that the user requirements are valid and define at least one configuration in the configuration space. The AAFM provides the *Valid Partial Configuration* operation that takes a FM and a set of user decisions as inputs, and returns a boolean value indicating if these decisions are consistent with the FM and therefore it is possible to find at least one product for such requirements.

Once the requirements are checked for validity, an optimisation problem can be defined and solved using the *Optimisation* operation. The optimisation operation takes as inputs a FM, a set of user decisions and an objective function and returns a configuration that satisfies the decisions and optimises the function. In our case, the objective function refers to the minimal `TotalCost` attribute value. Figure §6.4.c shows our proposed approach.

6.7 Implementation and Verification

In this section, we describe the implementation and verification details of our approach. First, present the implementation of the analysis operations of Section §6.6. Second, we briefly motivate and describe the web scraper we employ to build the EC2 FM. And finally, we detail the implementation and verification of the EC2 FM.

6.7.1 Analysis Operations Implementation

The analysis operations proposed in Section §6.6 are implemented in the *FaMa Framework*. FaMa Framework[4] [158] is an AAFM open-source tool that supports more

[4] www.isa.us.es/fama

than 20 different analysis operations with 4 different reasoners. What we had to develop was the proposed transformation from attributed FMs to basic FMs in Section §6.6, in order to obtain the number of configurations of an IaaS FM.

FaMa employs different logical paradigms, such as propositional logic or CSP, to perform the AAFM. This means that the FM is translated to logic variables and constraints, which are interpreted and resolved by off-the-shelf solvers. Features and attributes are mapped as variables, while relationships and constraints are mapped as different kinds of constraints. Depending on the AAFM operation, additional inputs can be mapped, and different operations are invoked on the solver. More details about the AAFM and its mappings, and details about the operations are available in [19].

However, the traditional mapping of the AAFM– and consequently FaMa – presents performance issues for specific operations, and especially for the Optimisation operation. For this reason, we have decided to add a new reasoner to FaMa, based on the CPLEX CP Optimizer solver [5], to tackle these performance problems. This new reasoner implements the Optimisation operation, and present three main changes:

- *Alternative FM mapping.* Traditionally, the default mapping to CSP that solvers proposed in the literature maps each feature to a boolean variable [18]. This is the mapping that all the FaMa reasoners adopt. In our alternative implementation, however, we have mapped each of the six configuration options of the EC2 FM as an integer variable. Each variable has as many values as leaf features the configuration option has.

- *CPLEX tuples.* By default, every FM constraint is translated to a solver constraint by the FaMa reasoners. Nonetheless, in models with thousands of constraints this may lead to an overhead in the solvers. For this kind of models, CPLEX provides tuples, which improve dramatically the performance by describing all the elements within a set one by one. Our alternative implementation represents the availability and pricing constraints using CPLEX tuples. We also modelled attributes internally as integers, since CPLEX tuples lack support for real numbers. Anyway, the impact in the accuracy of the total cost is negligible, as Table §6.8 shows.

- *Search strategy.* We use a customised search strategy, defining the configuration options of EC2 as the decision variables of the problem.

[5] http://www-01.ibm.com/software/commerce/optimization/cplex-cp-optimizer/

6.7.2 EC2 Web Scraper

As stated in Section §6.5, the EC2 FM presents more than 20,000 constraints. Most of such constraints determine the exact cost hour of each configuration, and the configurations that are not available for different reasons. The constraints are presented by AWS in tables in its EC2 website, but its number make its manual processing tedious and error-prone.

The retrieval of the pricing and availability constraints from Amazon EC2 can be considered as an additional challenge. There is no standard interface to retrieve this information directly from the provider. This requires the use of automated techniques, such as web scraping [124] on the provider's website, to get the data. Although we perform this technique to build the model, it is an implementation aspect to verify our proposal – it is a means to our end but not the end itself. Another issue related to the information retrieval is the evolution or variation of the configuration space. This happens often in commercial cloud services, and requires manual work to adjust the scraper and or update manually the model. We are aware of this limitation, and therefore in this paper we work with a snapshot of Amazon EC2 from 12th June 2014.

We have developed a web scraper to automatically extract the constraints from the EC2 website. We have to remark that this is a basic and non-adaptable scraper, i.e. it is not intended to adapt itself for the changes and evolution of the EC2 website, but to automate the constraints extraction. The scraper is developed in Java using the jSoup library [6] and takes as input the FM tree and attributes, and their correspondence with the elements in the EC2 website. It processes the characteristics and pricing tables, and produces as output the characteristics of each instance type – RAM, ecu and cores – and the pricing tables of EC2 as FM constraints in the FaMa plain text format – see next subsection for this format. In the case that a configuration does not appear in the pricing tables it means that is unavailable, so the scraper generates a constraint to remove such configuration from the EC2 FM.

This scraper only extracts instance characteristics, pricing and availability constraints. Changes in the instance pricing or availability are supported by our EC2 scraper. However, any change in the configuration options or their values, or in the presentation or format of the EC2 website would require manual fixes on the scraper. Indeed, during the realisation of this paper, we have made adjustments on the scraper two times due to changes in the website format and the addition of new instances.

[6] http://jsoup.org/

6.7.3 EC2 FM

Although there are several formats to define FMs, we have chosen the FaMa plain text format [84, 158]. The most well known notation is the so-named feature diagram, employed for the excerpt of Figure §6.3. This format is adequate for illustration purposes and simple FMs. However, in our case, our EC2 FM is large and complex enough to use a text format. There are different alternatives for FM text formats [53], but only a few of them provide automated analysis support. Since we take advantage of the analysis operations already provided by FaMa, we use the FaMa notation to define the EC2 FM in plain text.

The proposed methodology to build an IaaS FM relies on a deep understanding of the service and on automated information retrieval mechanisms. While in this paper we have implemented a basic scraper for the latter, some errors may appear due to the former during the definition of the tree, the attributes or some constraints. Consequently, we find useful to employ some of the analysis operations of Section §6.6 to verify the model. On the one side, we want to calculate the completeness and correctness of the EC2 FM with respect to the retrieved configuration space information. On the other side, we want to ensure that the EC2 FM is free of FM errors, as described also in Section §6.6. Note that finding errors in the FM may not imply that there are errors on the building on the model, but on the configuration space of the service.

Completeness and Correctness

We need to ensure that the EC2 FM gathers all the available EC2 configurations, but no more. That is, we need to check the completeness and correctness of the EC2 FM. For this purpose, we employ several of the operations explained in Section §6.6 and Figure §6.4. First, we parse the tables of the EC2 website to get the EC2 available configurations. Then, we check that each and every retrieved configuration is contained in the EC2 FM, by means of the Valid Configuration operation. In this way, we measure the completeness of the EC2 FM. If all the valid EC2 configurations are represented in the FM, we get the number of configurations of the FM, and compare with the number of configurations retrieved from the tables. If the number is the same, we can say that our EC2 FM is correct and complete.

The EC2 FM required several iterations to correctly represent the configuration space. Initially, the model represented several spurious configurations, due to some unavailable configurations are just excluded from the EC2 pricing tables instead of be marked as “N/A”. After three rounds of fixes, we made the necessary changes to

Expressiveness Comparison			
	EC2 FM	**AWS TCO**	**CS**
Multi-provider	No	No	Yes
OS	Generic & specific	Linux	Generic
Location	State & Continent	AWS areas	Area
Dedication	Yes	No	No
Usage data	hours/ month	% of 3 years	4 values
Period	1 to 48 months	% of 3 years	1, 3, 6, 12, 24, 36 months
RAM	GB	GB	GB
Comp. unit	ecu, cores	processors, cores	CS units
Storage	GB	GB	GB

Table 6.5: Expressiveness comparison among EC2 FM, Amazon TCO and CloudScreener.

remove the fake configurations and consider all the available ones.

Error Checking

In this step, we verify, using FaMa and its analysis operations, the validity of the EC2 FM. As stated in Section §6.6.1, the error checking operation detects behaviours that suggest a wrong modelling, or errors in the configuration space. The execution of this operation in FaMa detects no errors.

6.8 Evaluation

In this section we compare, in terms of optimal cost and expressiveness, the results obtained by our approach with the results obtained in commercial tools, in particular CloudScreener (CS) and Amazon TCO. Besides, we present an experimental study to check the performance of our prototype.

6.8.1 Comparison to Other Approaches

Our first comparison study shows that, in terms of expressiveness, our EC2 FM provides the users more freedom to express decisions than two "competitors": AWS TCO tool and CloudScreener. AWS TCO [3] is a tool provided by Amazon to "compare

Comparison of Results									
			ECU/		Cloud Screener			Our approach	
	OS	Area	CSPU	RAM	Config	Cost/h	Reqs.	Config	Cost/h
C1	Linux	US East (VA)	12/6	8	m3.large	0.14 $	No	m3.xlarge	0.28 $
C2	Linux	South Am.	4/2	9	m3.xlarge	0.381 $	Yes	m2.xlarge	0.323 $
C3	Win	Singapore	8/4	40	hs1.8xlarge	5.901 $	Yes	r3.2xlarge	1.292 $
C4	Linux	US West (CA)	68/34	160	r3.8xlarge	3.12 $	Yes	r3.8xlarge	3.12 $
C5	Win	Europe	16/8	5	c3.xlarge	0.376 $	No	c3.2xlarge	0.752 $

Table 6.6: Comparison of results with CloudScreener. m3.large instances present 7.5 GB of RAM, while c3.xlarge instances present 14 ECU.

the cost of running your applications in an on-premises or collocation environment to AWS". CloudScreener [38] is a web application to "*easily identify your need in terms of infrastructure and compare available offers upon many criteria*". While both of them provide web interfaces, we use the EC2 FM for expressing user requirements. Table §6.5 shows a comparison about the main elements of the three approaches. Except for choosing among different providers, our approach presents more configuration options for the measured configuration aspects. For instance, AWS TCO ignores all the OS but Linux. About CloudScreener, for example, they only offer four fixed values to express hours per month information.

The second comparison study, about the optimality of the results, shows how our analysis approach improves the results of CloudScreener, which sometimes even violates the user requirements. For this comparison, we have taken 5 test cases. We have used them as input for our analysis, and also for the web application of CloudScreener using the "*expert mode*". The inputs and results of the comparison are shown in Table §6.6, and a record of the process is also available [7]. In order to facilitate the comparison, we have even set as constant the following properties: Storage ≥ 0, period = 1, usage = 24 h/day = 730 hours/month, and dedication = public, since CloudScreener ignores dedicated instances. CloudScreener uses an own unit, CSPU, to define the computing power, as "an equivalent to EC2 unit (ECU)". Therefore, we divide by 2 the ECU of our inputs when translating to CloudScreener. As Table §6.6 shows, our approach improves CloudScreener results for four of the five test cases. CloudScreener results for C1 and C5 are not even valid, since they violate customer requirements – RAM for C1 and ECU for C5 –. For C2, and specially for C3, our analysis improves the

[7] http://youtu.be/apQmFV5ilUA

instance selection, obtaining a cheaper instance that still satisfies user needs.

Experimental settings									
Cases	*ECU*	*RAM*	*Disk*	*Usage*	*Period*	*SSD*	*OS*	*Loc.*	*Dedicated*
Group 1	[1,20]	[1,10]	[1, 1 000]	[1, 730]	[1, 36]	bool	enum	enum	bool
Group 2	[6,60]	[10,40]							
Group 3	[20,108]	[40,244]	[1 000, 10 000]						

Table 6.7: Experimental settings groups and ranges.

6.8.2 Performance Study

Although the migration is not a real-time task, we want to ensure that our approach performs the analysis in a reasonable time.

With the intention of testing our analysis approach and its performance, we have implemented an automated generator of user requirements for EC2. This generator produces decisions about the main configuration options described at the end of Section §6.5, that is, location, OS, dedication, instance characteristics and usage data. Most of the decisions are randomised on the domain of the configuration option, as Table §6.7 shows. For instance, location value may be a specific area like Virginia, or a group of areas like NorthAmerica. In contrast, we have managed in a different way RAM, ECU and storage values. Such properties have large domains, but instance types are distributed exponentially along the domain, as Figure §6.5 shows with its logarithmic scale. Consequently, we have defined three different groups of test cases, in order to aim for areas where different instances can satisfy customer requirements. We have generated 200 test cases for area, for a total of 600 test cases.

Both analysis approaches – FaMa and CPLEX-based reasoner – produce the same output configurations, but in terms of performance, our CPLEX-based reasoner outperforms FaMa up to several orders of magnitude. While FaMa lasts for minutes in most of the cases, the alternative implementation outputs the same result in less than a second. As shown in Table §6.8, the average difference in the analysis time is three orders of magnitude (10^3), while the standard deviation follows the same scale. Total cost errors, due to the round of real variables to integer variables, are negligible, since average and standard deviation values are around cents of dollars.

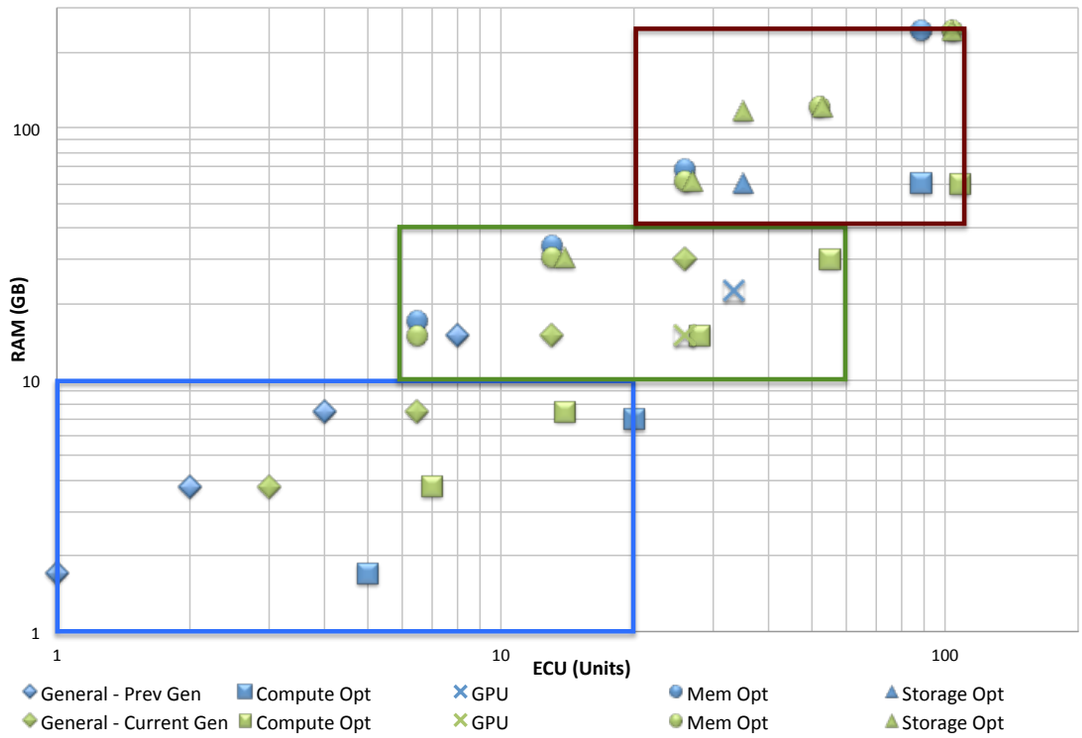


Figure 6.5: Instance types distribution in terms of RAM and ECU (log10 scale). Each coloured rectangle denotes a different area of test cases.

Implementation Comparison				
	Performance (ms)		**Total cost diff.($)**	
Impl.	Avg. (μ)	SD (σ)	Avg. (μ)	SD (σ)
FaMa	1.73×10^5	1.9×10^5	0.4	0.33
Alternative	5.19×10^2	2.15×10^2		

Table 6.8: Comparison between FaMa based impl and our alternative impl in terms of performance and output

6.9 RELATED WORK

6.9.1 Cloud Migration

Jamshidi *et al.* [86] present a systematic literature review of existing research works (up to 23) about planning, executing and validating migration of legacy systems towards cloud-based software. The authors identify a reference model for the migration

process, where they classify the different works depending on their scope. They also identify a general lack of tool support for the migration process, which we try to palliate in this paper. Frey et al. [55, 57, 58] propose an overall migration approach, based on the concept of CDO (Cloud Deployment Option). A CDO represents "a combination of a specific cloud environment, deployment architecture, and runtime reconfiguration rules for dynamic resource scaling". This approach provides CDOSim [55], a simulator to evaluate CDOs costs and response times, and CDOXplorer [58], an evolutionary algorithm to search for well-suited CDOs in terms of response time, cost and SLA violations. Both approaches are integrated in CloudMig [57], a suite to assist the migration of applications to the cloud. Differently from our work, which is focused on infrastructure migration, they are concerned about migrating applications. Khajeh-Hosseini et al. [90, 91] describe the challenges the users face when they plan to adopt the cloud. They introduce the cloud adoption toolkit to tackle such challenges, which provides a framework to assist the users in the decision process. This tool provides support to analyse, among others, technology suitability, energy consumption and cost prediction of providers and configurations. However, they provide exhaustive information rather than automating the search for the most suitable configuration. CloudGenius framework, an approach by Menzel and Ranjan [109], provides a process and decision support for migrating to the cloud. Their approach comprises a formal mathematical model and a migration process, whose goal is to lead to a VM image and cloud infrastructure selections. However, the tool support is still in a preliminary stage. Beserra et al. [24] present CloudStep, a decision process to support the migration of legacy applications to the cloud. CloudStep relies on template-based profiles of the migrating company, its legacy application and the candidate providers. Such profiles are cross analysed to identify and solve constraints, and to create a migration strategy. Kwon and Tilevich [93] propose an automated transitioning for applications to use cloud-based services. This approach can be seen as a crosscutting concern of the migration to the cloud, and in particular of our proposal to assist the configuration.

There is a number of works that focus on algorithms to automatically evaluate or rank cloud services configurations. Truong and Dustdar [164] present a service for estimating, monitoring and analysing costs associated with scientific applications in the cloud. They rely on cost models and experiment with real-world applications. Trummer et al. [163] interpret the outsourcing of part of the IT-stack as a constraint optimisation problem, that they solve using existing solvers. Tsai et al. [165] consider a similar approach, choosing between different cloud providers using data mining and trend analysis techniques, and looking for minimising cost. In a related research, Sun-

dareswaran et al [150] propose indexing and ranking cloud providers using a set of algorithms based on user preferences. Venticinque et al. [167] describe an approach to collect cloud resources from different providers that continuously meet requirements of user applications. The related work of Borgetto et al. [26] is oriented to software reallocation in different virtual machines in order to decrease energy consumption.

Some other works propose processes and methodologies to assist the migration, and identify factors and tasks. The work of Mohagheghi and Sather [114] presents some software engineering challenges related to migrating legacy systems to cloud services. They identify application modernization and understanding cloud technologies as the main issues. Tak et al. [151] identify cost key factors when migrating to the cloud. Different deployment alternatives, based in Azure and Amazon services, are benchmarked and detailed in terms of cost. Lloyd et al. [97] discuss which factors should be accounted when deploying to a cloud. The authors focus on bottlenecks which appear when scaling applications, and the impact of provisioning variation. Zardari and Bahsoon [179] rely on goal oriented requirements engineering (GORE) and specific tasks to assist users in the adoption of cloud services.

6.9.2 Variability, Ontologies and Cloud Services

Applying Software Product Lines and variability techniques to cloud services is attracting attention. Quinton et al. [128] propose a software product lines based approach that supports stakeholders while configuring a cloud environment and automates the deployment of such configurations. They also use FMs for the configuration process, but while they consider features as deployable artefacts of cloud environments, we consider features as configuration points and specific values of commercial cloud services. In the same line, Schroeter et al. [144] use FMs to configure IaaS, PaaS and SaaS, and also present a process to manage the configuration of several stakeholders at the same time. Dougherty et al. [49] also uses FMs to model IaaS, but the goal in this case is reducing energy cost and energy consumption, towards the development of a "green cloud". In a different way, Cavalcante et al. [32] proposes the extension of traditional software product lines with cloud computing aspects. Our work differs from these approaches in a key way: while they focus on deployable cloud environments, and consider features as deployable artefacts, we focus on commercial cloud services, and consider features as configuration points and specific values of the services. Wittern et al. [176, 177] present the so-named service feature modelling, for the representation of service design concerns, and to capture their potential combinations.

They also also provides a method [176] to rank service design alternatives based on stakeholder preferences. While they focus on capturing diverse aspects of cloud services in a general way, we focus on the complete modelling and analysis of a particular service, EC2, and compare our approach against commercial applications.

The use of ontologies has been proposed by several authors to assist the modelling and selection of cloud services. Han et al. [78] construct a cloud ontology consisting of a taxonomy of concepts of Cloud services, in order to support a cloud service discovery system. Dastjerdi et al. [45] propose an ontology-based discovery to provide QoS aware deployment of appliances on Cloud service providers. Ngan et al. [121] also present a semantic cloud service discovery and selection system, based on OWL-S. The proposed system supports dynamic semantic matching of cloud services described with complex constraints. Garcia-Rodriguez et al. [134] propose a semantically-enhanced platform to assist the process of discovering the cloud services, stored in a semantic repository, that best match user needs. While ontologies are an excellent option for a modelling approach, they present performance issues and reasoning lacks on integer and real variables, which are inherent to the problem we tackle in this paper.

6.9.3 Commercial Approaches

Configuring and analysing cloud platforms/providers is continuing to receive significant attention also from the business community. Besides previously referred CloudScreener [38], there are a number of companies and start-ups providing comparisons, benchmarks and configuration support for cloud services. Cloudorado [37] also provides a cloud services price comparison engine. CloudHarmony [36] is a startup which looks for obtaining metrics about cloud providers performance, and provides a comparison framework for many services providers. PlanForCloud [126] is another startup, focused on configuring and simulating cost of several cloud platforms, like Amazon, Azure or Rackspace. They provide interesting options, like creating elastic demand patterns, and filtering by options like OS or computing needs.

As shown in the evaluation section, our proposal improves the expressiveness and configuration results returned by some of these approaches. In particular, CloudScreener returned false positives for the search of the best configuration that we improved with our analysis. However, we still lack cross-provider support. In future works, where we will consider different cloud providers and metrics, we may extend our evaluation with some of these approaches.

6.10 SUMMARY

We have presented an approach to assist the configuration selection when migrating an in-house computing infrastructure to the cloud. In terms of expressiveness, our modelling approach provides greater degree of freedom for the customer when making decisions about such configuration, compared to commercial applications like AWS TCO and CloudScreener. In terms of accuracy, we have proved that our analysis approach improves results compared to CloudScreener, which sometimes even violates user requirements. In terms of performance, we have implemented an analysis prototype to obtain the most suitable configuration whose execution times are negligible, being most of the times less than a second.

For such purposes, we have presented a modelling methodology to describe an IaaS as a FM. Modelling the configuration space of a large cloud provider, such as AWS, as a FM provides a well-structured and compact representation to express user infrastructure requirements. The resulting model still contains, for the particular case of EC2, lots of information, with a huge number of constraints – more than 20,000. The addition of abstract features and attributes eases the decision making, since information like RAM, period of commitment or total storage is closer to the user than specific instance types, purchasing options or EBS extra storage.

We have also proposed the use of the AAFM to assist the configuration when migrating to an IaaS. With the support of the AAFM operations, user requirements can be validated and employed to obtain suitable EC2 configurations. From the variability management perspective, we have (1) demonstrated the applicability of the AAFM for the configuration cloud services, (2) defined the product listing for attributed FMs, and (3) proved that off-the-shelf AAFM mappings, such as the one provided by FaMa framework, can be improved to dramatically reduce execution times.

Our approach can be generalised to other scenarios – including those considered in this work, primarily focusing on a migration to the cloud. In a recent work [71], we propose the adaptation of a multi-tenant service to satisfy the changing needs of the users. In this sense, our analysis approach could be applied to IaaS reconfiguration scenarios where the user needs to change and the current configuration becomes unsuitable. Our analysis approach could also be integrated with existing tools. An integration, for instance with the AWS calculator, would provide us with a reliable way to get the exact pricing of EC2, and it would provide the calculator with capabilities for the automated search of the most suitable configuration for given needs. The

modelling methodology has also the potential for a cross-provider analysis. However, in this case, we would have to propose a common IaaS FM, and also address issues related to *ontology alignment* between the concepts of different providers.

However, our approach still has limitations and room for improvement, that may motivate new works in this area. In addition to a cross-provider analysis, we think that FMs and our modelling methodology could be employed for different cloud services. We also consider defining a metamodel for the configuration of cloud services, based on FMs, but with specific mechanisms to get beyond some limitations of FMs and the current language. For instance, there should be a better and more compact way to define price policies and temporal-aware requirements [102, 116]. It is also necessary to enable a way to express constraints over several instances of the same services, such as the discount policies for multiple hired services, which is a current limitation of FMs. On the customer's side, FMs lack mechanisms to express fuzzy needs or preferences [63] as well as requirements. An extension would provide users with greater capability to specify their needs. The aforementioned AWS calculator could be used also to verify that we have described the EC2 pricing policy correctly. Finally, we think that we can derive additional benefits from the AAFM. It is necessary to analyse in depth the applicability of all the existing operations of AAFM to the services field.

User-centric Adaptation Analysis of Multi-tenant Services

Everything changes and nothing stands still.

Heraclitus (535 - 475 BC),
Philosopher

In this chapter, we present the content of the paper submitted (and currently in the second round of revision) to the Transactions of Autonomous and Adaptive Systems of ACM. In Section §7.1 we introduce the paper, while Section §7.2 provides some background on multi-tenancy, DaaS and feature modelling. Section §7.3 illustrates our DaaS case study and Section §7.4 introduces the user-centric adaptation problem and, in particular, the adaptation analysis. Section §7.5 describes how multi-tenant services and preferences are modelled to support the analysis, which in turn is presented in Section §7.6. Section §7.7 discusses our experimental results and Section §7.8 points out the open issues that will be addressed in future work. Section §7.9 compares our approach with relevant related work, and finally Section §7.10 concludes.

7.1 Introduction

Multi-tenancy allows cloud providers to deliver the same service to different customers, which share physical and/or virtual resources transparently [25, 120]. Depending on the adopted cloud service model, users 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 bigger the resources and cost savings, but the smaller the configurability. Limited configurability [113] is a major drawback, especially when user preferences are not known in advance. Several approaches [13, 92, 113, 143, 144] 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 [2] considers changes in the user collective judgement as a new adaptation driver. Other approaches [30, 98] consider conflicting users preferences and limited infrastructural resources in the construction of adaptive software systems. These approaches identify a system architecture [98] or configure a service-oriented application that maximises QoS preferences [30] 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 paper, we characterise the user-centric adaptation of multi-tenant services problem, focusing on the adaptation analysis. In our previous work [71] we highlighted the challenges to be addressed for engineering the activities of the MAPE (Monitoring, Analysis, Planning, Execution) loop [89] necessary to support user-centric adaptation and proposed a preference-based analysis that maximises in a balanced way the satisfaction of user preferences, which are expressed on the possible service config-

urations. In this paper 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 the users or the operational environment (including available service configurations and infrastructural resources) change at runtime.

We illustrate and motivate our approach by utilising different cases of virtualised desktops. Compared to our previous proposal, which focused on a hosted shared DaaS, this paper 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 as configuration space, the infrastructural resources and the workload by using FMs [88]. We represent the user preferences by adopting an existing preference model [65]. Our adaptation analysis is interpreted as a multi-objective constrained optimisation problem built on top of the AAFM [19]. The optimisation problem is solved by using metaheuristic algorithms [99], which have been proved suitable for FM optimisation in existing work [76, 140]. 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 as 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 paper is organised as follows. Section §7.2 provides some background on multi-tenancy, DaaS and feature modelling. Section §7.3 illustrates our DaaS case study and Section §7.4 introduces the user-centric adaptation problem and, in particular, the adaptation analysis. Section §7.5 describes how multi-tenant services and preferences are modelled to support the analysis, which in turn is presented in Section §7.6. Section §7.7 discusses our experimental results and Section §7.8 points out the open issues that will be addressed in future work. Section §7.9 compares our approach with relevant related work, and finally Section §7.10 concludes.

7.2 Background

This section provides some background on multi-tenancy, Desktop as a Service delivery models, and feature models.

7.2.1 Multi-tenancy

Multi-tenancy is defined as *multiple customers, organizations or processes (tenants) sharing common physical or virtual computing resources while remaining logically independent* [120]. Typically, a tenant *groups a number of users, which are the stakeholders in the organization* [25]. Shared resources can vary depending on the cloud service model; each model provides resources belonging to different levels of abstraction. The 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 PaaS model, providers deliver a computing platform, typically including 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 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 paper we focus on the case of multi-tenant DaaS relying on the delivery models provided by Citrix [33].

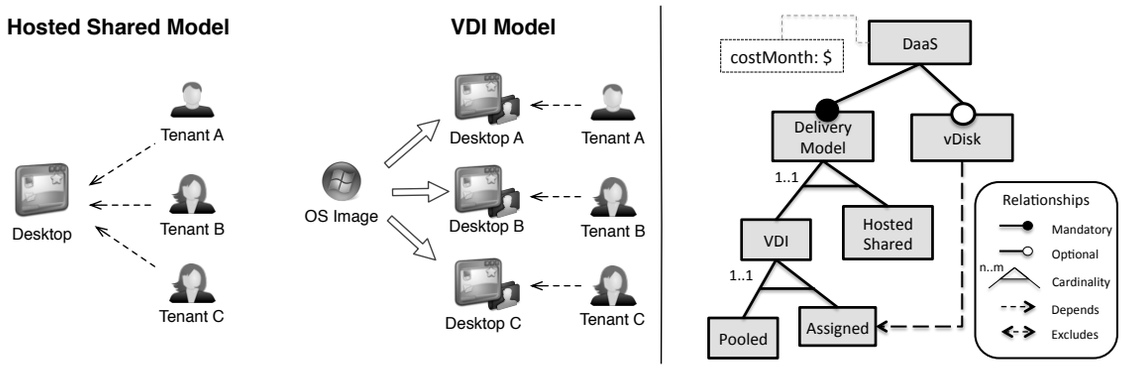
7.2.2 Desktop as a Service Delivery Models

DaaS delivery models differ depending on the specific provider. In particular, as shown in Figure §7.1a, 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 server based operating system. In addition, because multiple users are sharing a single operating system, they are prevented from performing actions that may negatively

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

affect other users, such as installing new applications or changing system settings.



(a) (b)

Figure 7.1: a) DaaS delivery models b) Example of a FM.

The VDI model hosts custom desktop instances on remote servers. Each desktop instance is associated with a different tenant and relies on a centralised master image. VDI supports more customisation 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) [33]. A pooled VDI provides a clean random virtual desktop each time a user accesses the service. An assigned VDI allows the users to customise 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 profile (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, while in the case of a VDI model, the shared aspects depend on the concrete VDI configuration. Table §7.1[4] 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. While 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.

[4]We assume to use Windows Server 2012 and Windows 8 for VDI and hosted shared models, respectively, and a processor speed of 2.7 GHz and Intel Westmere processor architecture.

Impact of user profiles						
		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 (1340)	13 (1546)	21 (957)	1 GB	340 MB
Normal	2-10 office apps. light multimedia use	11 (1827)	10 (2010)	14 (1435)	2 GB	500 MB
Heavy	Multimedia or app development	6 (3350)	5 (4020)	7 (2871)	4 GB	1 GB

Table 7.1: Impact of the user profiles on the required infrastructural resources for different DaaS delivery models [33].

7.2.3 Feature Models

Feature Models (FMs) [88] are used to represent all the possible products that can be built in variability-intensive systems such as 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 cross-tree relationships. These relationships define how features can be combined in a product, defining the configuration space of the system. Figure §7.1b shows an example of a 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 a FM, non-leaf features [153] 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.

7.3 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 a 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, where each tenant groups users having compatible DaaS configuration preferences.

7.3.1 DaaS Configuration Space

We present a Windows-based DaaS example where 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 §7.2 indicates the 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; while in an assigned VDI model such option can be customised for each tenant, avoiding any possible conflict.

7.3.2 Infrastructural Constraints

In this section we characterise 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 s/he executes more often. We profile users along the three categories identified by [33]: light, normal and heavy. The delivery model has an impact on the number of users per core a DaaS is able to handle and on the required RAM size. Table §7.1 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)*

DaaS configuration options							
Shared options						**Workload Peaks**	
HS	PVDI	AVDI	**DaaS configuration options**		**Values**	**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.

Table 7.2: Shared DaaS configuration options depending on the delivery model and their workload peaks.

and memory size [5]. Beside 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 §7.2. Note that a peak workload represents the maximum workload that can be reached at a given time instant.

7.3.3 Users, Preferences and Conflicts

In our example scenario we assume to have 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 §7.3. 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

[5] While 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) 3.3 Ghz processor. We have to adjust the clock frequency to the Westmere 2.7 Ghz of Table §7.1. The MIPS for a single core are $\frac{147{,}600*2.7}{6*3.3} \approx 20100$.

Tenants profile and preferences				
	# 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

Table 7.3: Tenants' usage profile and preferences. Potential conflicting preferences are associated with the same number.

OS updates, while tenant 3 favours monthly office updates causing a potential conflict. Indeed, the satisfaction of both preferences may cause a violation of the constraint that requires that office updates period must be smaller than the OS update period (Table §7.2). Note that the complete satisfaction of all the preferences is infeasible in most of the 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 therefore they join and leave the DaaS almost at the same time in the workdays. While the users of 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 sub-optimal because tenant preferences vary during the system life-time. 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 of the available service configurations might lead to changes in the tenant preferences. For example, if some of the backup features are removed, the users might change their preferences w.r.t. the new configuration space.

7.4 Towards 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.

7.4.1 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 maximise 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 the users 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 the system usability. Therefore it is necessary to balance the trade-off between the preferences satisfaction and the obtrusiveness of the adaptation actions.

As shown in Figure §7.2, the activities of the MAPE loop can support user-centric adaptation as follows:

(a) **Monitoring (M)** has the objective to capture 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 of 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 users feedback [2]. 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 the 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 the users [148] and, therefore, preclude the execution of a reconfiguration.

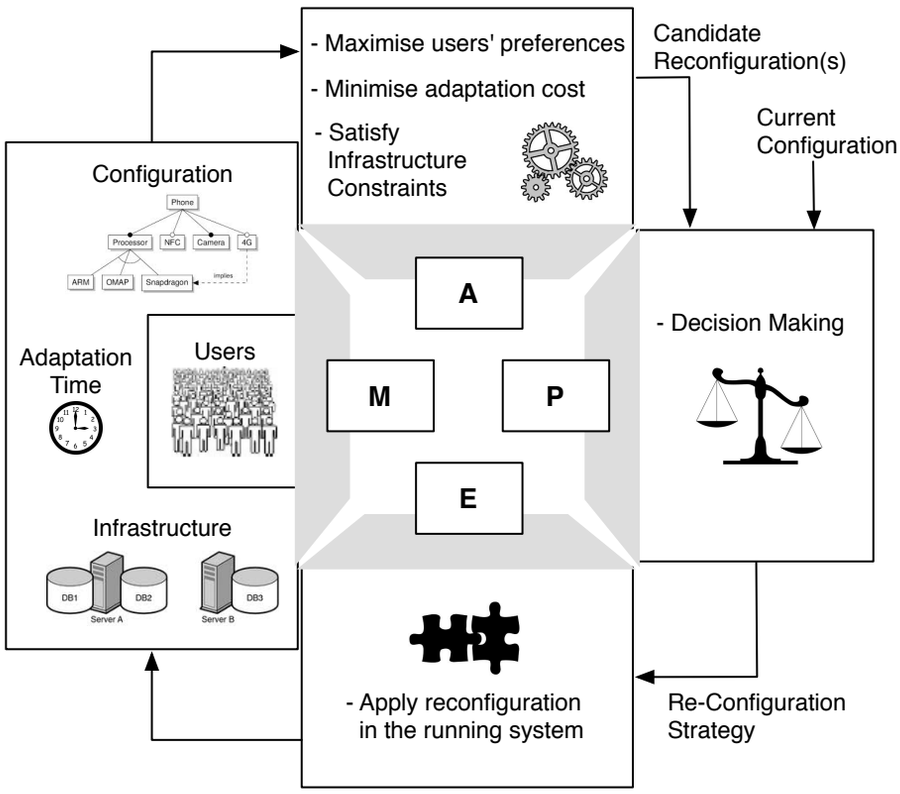


Figure 7.2: User-Centric Adaptation MAPE (Monitoring, Analysis, Planning, Execution) Loop.

(b) **Analysis (A)** has the objective to identify the best system configuration(s), which optimises a set of metrics. In particular, a reconfiguration should maximise the satisfaction of the 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 [59], the reconfiguration should also minimise 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 the obtrusiveness. This issue is further discussed and parametrised in sections §7.5.1 and §7.6.3, 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 the 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 to apply 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 [13, 144]. While, 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 the 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 organisation.

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[6], which is an open source blogging tool and a content management system providing different customisation options and plugins. Similarly to a DaaS, Wordpress supports multisites[7], 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. Virtualised 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.

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

7.4.2 Adaptation Analysis for Service Reconfiguration

In this paper 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 characterises the infrastructural resources; T represents the set of tenants, $f_C : C \to I$ and $f_T : T \to I$ are the functions that identify the impact (workload) that each configuration and each tenant has on the required infrastructural resources, respectively; $u_i : C \to \mathbb{R}$ are utility functions for each tenant in T defining the tenant satisfaction for any given configuration in C; and $\rho : C \times C \to \mathbb{R}$ 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 $U : C \to \mathbb{R}$ 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.

7.5 MODELLING

In this section we describe how the multi-tenant service (Section §7.5.1) and the user preferences (Section §7.5.2) are modelled 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 SOUP preference model [65] to represent the user preferences.

7.5.1 Service Modelling

Service modelling usually involves multiple and interrelated configuration options. For example, Amazon EC2 features present more than 20,000 constraints defining 16,991 different configurations [69]. Our choice to use FMs to model multi-tenant services is

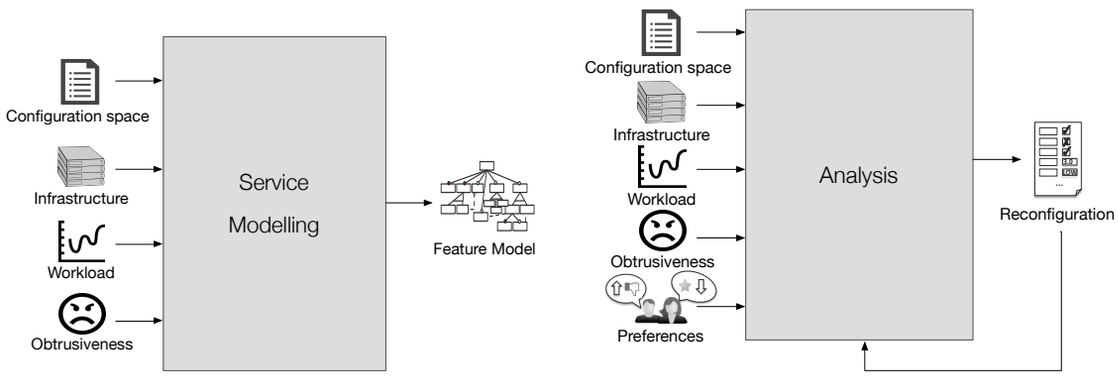


Figure 7.3: a) Service modelling. b) Analysis inputs and outputs.

motivated by the fact that FMs are expressive enough to represent increasingly complex systems such as cloud services [69], and content-management frameworks [139]. This section leverages our DaaS case study to describe how we use FMs to model multi-tenant services.

Configuration Space

Figure §7.4 shows a FM for the DaaS case study. Each configuration option is modelled 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 sub-features 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 of them 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, while 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.

Infrastructure and Workload

In real-world contexts, services have limited infrastructural resources which 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

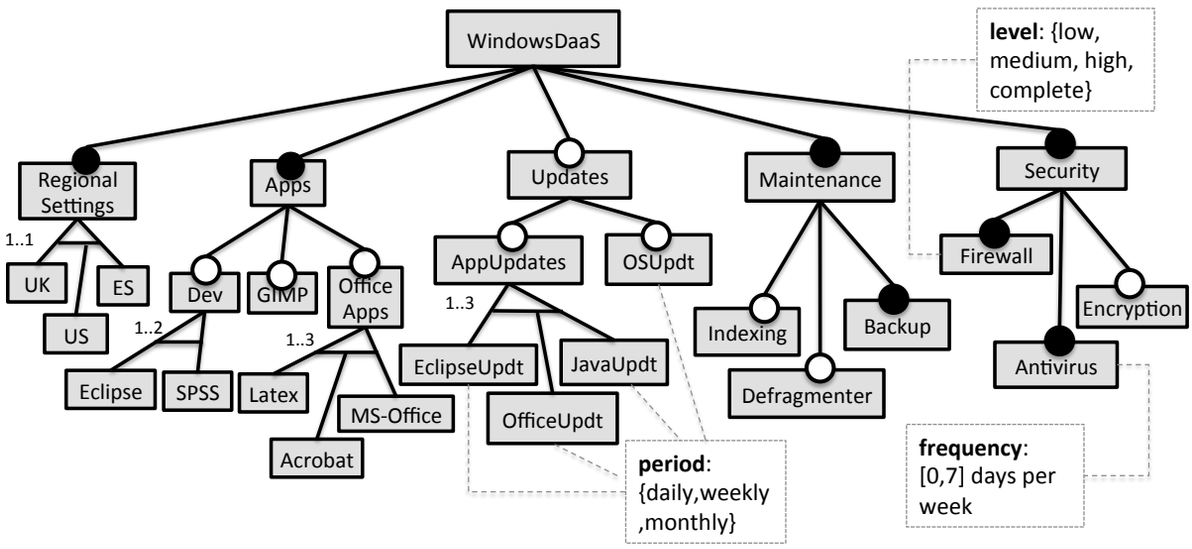


Figure 7.4: DaaS configuration space expressed as a Feature Diagram – a FM graphical notation.

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 7.1 shows an excerpt of the infrastructure and workload definition in the FM using the FaMa plain text notation [158]. 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 7.1). 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 the 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 7.1). 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 7.1, 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 non-linear, 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 paper is on the analysis phase, we use simulated workloads for the example described in Section §7.6.2.

Listing 7.1: Excerpt of DaaS infrastructure, workload and obtrusiveness modelling for memory using FaMa plain text format

```
%Attributes                                                    1
# Available infrastructure and workload attributes             2
WindowsDaaS.availableMemory: Real [256]; ## in GBs             3
WindowsDaaS.availableCPU: Real [400000.0]; ## in MIPS          4
WindowsDaaS.memoryWorkload: Real [0 to 512];                   5
WindowsDaaS.CPUWorkload: Real [0 to 1000000.0];                6
                                                               7
# Attributes to compute the current memory workload            8
WindowsDaaS.userMemoryWL: Real [0 to 256];                     9
WindowsDaaS.configMemoryWL Real [0 to 256];                    10
Weather.configMemoryWL: Real [0.03],0;                         11
Indexing.configMemoryWL: Real [0.5],0;                         12
...                                                            13
# Attributes to compute the obtrusiveness                      14
Gadgets.obtrusiveness: [3]; ## high obtrusiveness value        15
Indexing.obtrusiveness: [2]; ## medium obtrusiveness           16
JavaUpdt.obtrusiveness : [ 1 ] ; ## low obtrusiveness          17
...                                                            18
%Constraints                                                   19
```

```
WindowsDaaS.CPUWorkload < WindowsDaaS.availableCPU;                    20
WindowsDaaS.memoryWorkload < WindowsDaaS.availableMemory;              21
WindowsDaaS.memoryWorkload == WindowsDaaS.configMemoryWL               22
                            + WindowsDaaS.userMemoryWL;                23
WindowsDaaS.configMemoryWL == Weather.configMemoryWL                   24
        + Calendar.configMemoryWL +  Defragmenter.configMemoryWL       25
        + Indexing.configMemoryWL + Backup.configMemoryWL + ...;       26
...                                                                    27
```

Obtrusiveness

To model the obtrusiveness of a service configuration we leverage the conceptual framework proposed by Ju and Leifer [87]. This framework determines the obtrusiveness level of each interaction of the system with the user by considering the attention dimension, i.e. 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 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, the 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 the 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 7.1 shows an example of the obtrusiveness definition for the `Gadgets` and `Indexing` features. In Section §7.6.3 we explain how these values are used to include obtrusiveness information in the adaptation analysis.

7.5.2 User Preferences Modelling

In FMs, users can describe their preferences in terms of hard requirements, where 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 [65] to express fuzzy user preferences on a given service. SOUP is a highly intuitive and expressive preference model, which 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 [29]. 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 maximising 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 minimising 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 "weekly" value.

In this way, the different users – grouped 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 [65]. 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.

7.6 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 §7.3b. The analysis problem is interpreted as an operation of the *Automated Analysis of Feature Models (AAFM)*, as described in Section §7.6.1. 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 optimisation (Section §7.6.2) and the obtrusiveness-aware optimisation (Section §7.6.3), respectively.

7.6.1 Automated Analysis of Feature Models

AAFM is a discipline that deals with "the automated extraction of information from FMs using automated mechanisms" [19]. We leverage existing mappings from FMs to logic paradigms and off-the-shelf solvers to to implement our adaptation analysis. In particular, in this paper we use the optimisation operation provided by the AAFM framework to perform our adaptation analysis. This operation takes a FM and an objective function as input, and returns the configuration fulfilling the criteria established by the function. To optimise the value of the attributes defined in the FMs, relative order preferences have been considered in previous work [9]. However, as far as we are aware, no approach has considered how to optimise fuzzy, high-level user preferences expressed in a similarly to those described in Section §7.5.2. Therefore, we have tailored the objective function of the optimisation operation of the AAFM framework to support our preference-based optimisation.

7.6.2 Preference-based Optimisation

We interpret our preference-based optimisation as a multi-objective constrained optimisation 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 §7.4 shows how preference satisfaction is computed for the analysis. Each preference defines a satisfaction degree p_{ij} comprised

Preferences satisfaction			
Preference	**Element**	**Satisfaction Measure**	**Example**
Favorites(f)	Feature	$f = selected \implies p_{ij} = 1$	*Favorites*(Indexing)
Dislikes(f)	Feature	$f = removed \implies p_{ij} = 1$	*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)

Table 7.4: Preferences satisfaction.

between 0 and 1, depending on the value of the element referred 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 need to rank them in order to choose a single one. Since we aim 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 [118]. We define a variation of the Nash product ($\prod_i u_i$), the so-named *Normalised Nash Product*, to compare 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 NNP value, we select the one minimising the resources usage. The rest of the section illustrates our analysis approach through the scenario presented in Section §7.3.

We consider a hosted shared delivery model, where all the resources are shared among the tenants, as shown in Figure §7.4. We also consider the preferences of each tenant, their number of users for two subsequent time instants (t and $t+1$) as shown in Table §7.5. At time t there are three tenants and the DaaS is running configuration c_1, described in Table §7.6, which provides the satisfaction u_i for each tenant i (Table §7.5).

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 sub-optimal. Therefore, the analysis is triggered, returning a new configuration, c_2 (Table §7.6), which delivers improved utility values $u_i(c_2)$. The most remarkable improvements are for $tenant_2$ and

Preferences reconfiguration scenario							
	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 7.5: Preferences reconfiguration scenario* for a hosted shared delivery model (changes in bold).

Sample configurations					
	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 7.6: Enabled features and attribute values for configurations c_1 and c_2.

$tenant_4$, 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 NNP value increasing from 2.2 to 9.1.

Workload impact							
		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**

Table 7.7: Estimation of the workload impact on the infrastructure.

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 [103]. 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 §7.1, 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 is calculated as the sum of the peak workloads (Table §7.2) of each selected configuration options indicated in the last row of Table §7.6. The total estimated workload is shown in Table §7.7, 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 7.1.

7.6.3 Obtrusiveness-aware Optimisation

We characterise 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 §7.6) returns the set $\{\texttt{Indexing}, \texttt{JavaUpdt}\}$. According to the model excerpt of Listing 7.1, $\rho_m = \texttt{Indexing.obtrusiveness} + \texttt{JavaUpdt.obtrusiveness} = 2 + 1 = 3$. 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.

7.7 EVALUATION

In this section we illustrate the evaluation of our approach. We describe the implementation of our analysis (Section §7.7.1) and explain the experiments we conducted to assess its effectiveness and performance (Section §7.7.2). Finally, we present and discuss our results (Section §7.7.3).

7.7.1 Implementation

We have implemented a prototype to perform our preference-based optimisation that uses *jMetal*, a Java-based metaheuristics framework to solve multi-objective optimisation problems [51]. jMetal provides a number of metaheuristic algorithms to compute a Pareto front of the problem. Among all the algorithms that jMetal provides, we have chosen two genetic algorithms which are widely used for the analysis of FMs [76, 140]. 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* [54] and *NSGAII* [46]. Al-

though both algorithms are elitist, NSGAII establishes different non-domination levels when ranking the – fixed sized – population, while FastPGA merges and ranks the previous and current generation into a single – and adaptive sized – population. Due to their complementarity we decided to compare the behaviour of the two algorithms for our analysis. Since the notation we used to describe the configuration space (FaMa plain text notation [158]) 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 which 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 [140]. We measure the violated constraints of a configuration using Choco[8], 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 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 by the NNP value (Section §7.6). If all the returned points of the Pareto front have a $NNP = 0$ – due to each $u_i = 0$, our analysis chooses the solution that maximises the average satisfaction of the tenants.

7.7.2 Experiments

Our goal in this experimentation is checking the effectiveness of our analysis. We compare the results obtained by using FastPGA and NSGAII with those obtained by 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 where 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 §7.5. We define a snapshot as the state of the tenants and their preferences for a specific time instant (t). For every snapshot, we run

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

Changes		
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 7.8: Amount of changes between two consecutive snapshots at $t-1$ and t.

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

Table 7.9: Characteristics of the FMs used for the experimental study.

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 of them is 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 §7.8) 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 a 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 snapshopt 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 §7.8. 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 version, and we have employed BeTTy [145], 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 §7.9 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 randomised 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 optimisation for FMs [140]. We consider $W_{TOTAL} = 200$, since such value is close to the maximum number of users supported by a single real hosted shared DaaS[9]. 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 – we have increased the default number of evaluations (25000) to $Evaluations = 100000$.

7.7.3 Experimental Results and Discussion

Table §7.10 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, it is necessary to take into account that the preferences of the different

[9]Using 2xE5-2470 2.3 GHz 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/

tenants may conflict most of the times, making infeasible to achieve full satisfaction for such cases. However, genetic algorithms clearly outperform the random search, especially w.r.t the achieved average satisfaction.

The percentage of improved reconfigurations – $\text{NNP}(C_t) > \text{NNP}(C_{t-1})$ – 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 the changes between consecutive system snapshots. The more the changes, the more the chances to decrease the satisfaction of the previous configuration, and the more the chances to find an improved reconfiguration. Besides, since we look for egalitarian solutions, our algorithms discard solutions that may have 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 optimisation problem we need to consider the constraints in order to return valid solutions. While 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 the our analysis can be performed at runtime.

Figure §7.5 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 by using FastPGA. For the first FM, the random search algorithm performs worse than the genetic algorithms, but better than its own behaviour 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.

7.8 Open Issues

In this paper 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 the activities of the MAPE loop they belong to.

Analysis results									
	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

Table 7.10: Results of the preference-based analysis for FastPGA, NSGAII and random search (RS) algorithms.

(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 – for instance 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 the initial prototype of our analysis. In future work, we will use a more precise and updated model of user behaviour 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

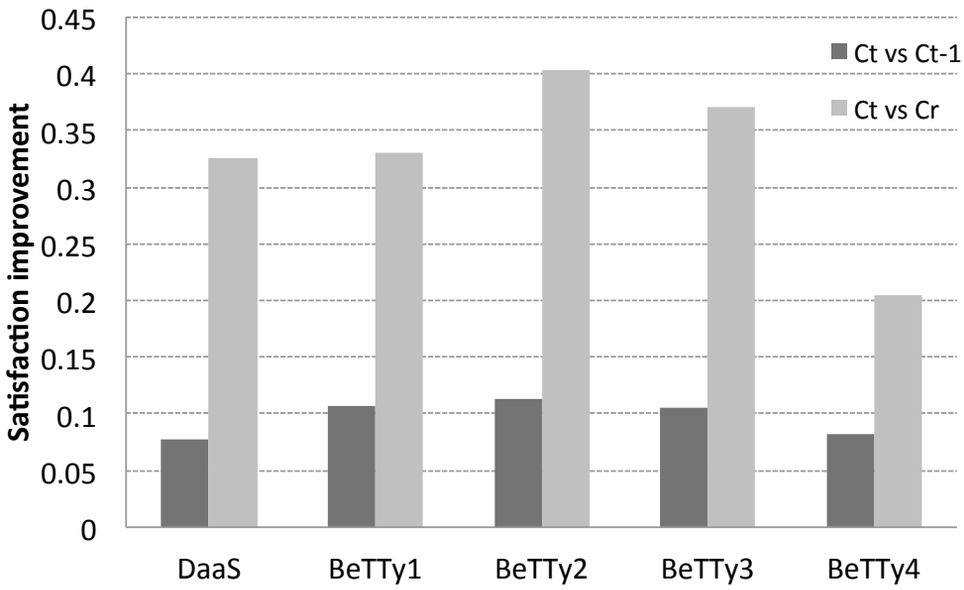


Figure 7.5: Average satisfaction improvement with respect to the previous configuration.

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 the reason why in our case study we did not allow the 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 the user satisfaction, 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.

7.9 RELATED WORK

The idea of performing adaptation to improve user satisfaction is not new. Other approaches [30, 98] have considered conflicting users preferences and limited infras-

tructural resources in the construction of adaptive software systems. For example- Malek et al. [98] propose the redeployment of software components on hardware nodes, in order to optimise conflicting QoS dimensions for changing user preferences. Cardellini et al. [30] present a reference framework for self-adaptation of service-oriented systems, where the user satisfaction is considered as an adaptation driver. Both approaches solve an optimisation problem to reconfigure a system architecture and a service oriented application, respectively. The optimisation problem is solved by means of different techniques, such as a integer programming or genetic algorithms. Ali et al. [2] propose social adaptation, which aims to dynamically adapt existing software systems depending on the user collective judgement 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. [44] propose to leverage the preferences of non-functional requirements expressed by a single user as a key driver for adaptation. Collected preferences are used to adapt the selection of routine tasks to be performed in a pervasive infrastructure by a user. Instead, we focus on the maximisation of the satisfaction of the global user preferences expressed on shared service configurations. Song et al. [147] present an approach to develop self-adaptive systems that takes into account end-users, who express their preferences on the adaptation actions selected by the system in order to better tune the adaptation results. Differently from this work, in our approach we consider user preferences as one of the main trigger for adaptation and not to improve the selection of a candidate configuration.

Lamparter et al. [94] propose an ontology for representing and matching configurable web services. In particular, service configurations and associated preferences are represented using function policies, which allow characterising 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. [62] who propose an algorithm to learn contex-

tual 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, while we assume that user preferences are given and address the problem of maximising their global satisfaction.

Cloud services analysis and adaptation has been a prolific research area during the last years. Caton and Rana [31] 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. [103] 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 in order to counteract SLA-violations. Wei et al. [171] 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. [125] also propose a resource allocation method which 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 [52] 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 to produce a better balance of utility and fairness. Differently from us, the approaches [31, 103, 125, 171] presented previously focus on resource allocation instead of feature selection and maximisation of the global user preferences. Furthermore, although the work by Wei et al. [171] and Pitt et al. [125] is also based on game theory to achieve fair resource allocation, their analysis has not ben proposed to support runtime adaptation. Finally, Vankeirsbilck et al. [166] propose a model for identifying an optimal resource allocation in order to satisfy virtual desktop requests based on the trade-off between costs and revenues for the service provider. The authors also consider the possibility of overbooking i.e. probability that less resources for the virtual desktops than needed are allocated. This approach is agnostic of user preferences and takes only into account resource allocation as a measure of SLA violations.

Other work is more focused on cloud adaptation at the application level. Inzinger et al. [83] propose a model-based adaptation which 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 preferences of multiple customers and low level infrastructural constraints. Marquezan et al. [100] provide a conceptual model that characterises 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 [119] 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 service 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 maximise the satisfaction of a single user. Our aim instead is to maximise global user' preferences and to minimise obtrusiveness of adaptation. A security oriented perspective is instead assumed by Bernal Bernabe et al. [22] who propose an advanced authorisation 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 paper. The idea of using variability techniques to model the adaptation space is not new. For example, Bencomo et al. [20] propose the use of variability modelling to define the runtime adaptation space. About multi-tenancy and SPLs, Schroeter et al. [143, 144] 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 [144], and also define requirements and middleware for a variable multi-tenant architecture [143]. 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. [113] propose to use variability modelling techniques to manage the variability of SaaS applications and their requirements. Specifically, they use variabil-

ity 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. [69], 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 user preferences. Similarly to us, Stein et al. [149] 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 adopted 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 optimisation in applications characterised by variability. Guo et al. [76] use a genetic algorithm to find optimal FM configurations for a single objective and user. Sayyad et al. [140] perform multi-objective optimisation of several large FMs using metaheuristics techniques. However, their objective functions are fixed (i.e. minimise errors and cost, or maximise number of features), while our fitness function depends on the specific user preferences. Finally, other work has explored techniques for solving conflicts in a configuration process. White et al. [174] propose a technique in this direction that only considers a single user and a minimal changes criterion. While, although García-Galán et al. [70] consider multiple users, after detecting the conflicts these users have to define explicitly the impact of every solution on their preferences satisfaction.

7.10 SUMMARY

In this paper, we present an approach to support user-centric adaptation of multi-tenant services. We motivate our proposal by using a multi-tenant DaaS case study and explain how to engineer the activities of the MAPE loop necessary to support user-centric adaptation. In this paper we focus on the analysis activity of the MAPE loop that identifies a service reconfiguration which maximises 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. The analysis takes as input the model of the service and the 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 service configurations. We adopt the SOUP preference model to represent user preferences.

The analysis is interpreted as a multi-objective constrained optimisation problem, where the different objectives are defined by the preferences of the tenants. This optimisation problem is solved by using genetic algorithms (FastPGA and NSGAII), which identify the Pareto front of potential candidate reconfigurations. Obtained experimental results demonstrate that our analysis approach 1) identifies reconfigurations that improve user satisfaction and 2) can be performed at runtime. FastPGA provides more effective results for smaller FMs, while NSGAII is more effective when bigger FMs have to be analysed.

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 the other activities of the MAPE loop. In particular, for the monitoring activity we will leverage existing work [62] to measure user preferences in a non-intrusive 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.

Part V

Final Considerations

8

Conclusions and Discussion

When you come out of the storm, you won't be the same person who walked in. That's what this storm's all about.

Kafka on the shore(2002),
Haruki Murakami

In this chapter, we present the conclusions of this dissertation. Section §8.1 summarises the conclusions, Section §8.2 discusses the controversial aspects and the future work, and Section §8.3 exposes the related publications.

8.1 Conclusions

In this thesis, we have enhanced and automated the support for the configuration of HCSs. Firstly, we have defined and clarified what configurable and highly-configurable services are. And in particular, we have proposed 1) a language to specify the decision space of HCSs and 2) a catalogue of analysis operations to assist the decision making for providers and consumers. In this sense, we have taken inspiration from the specification and automated analysis of BPMs [28, 47], SLAs [101, 115, 138] and variability [17, 60, 154]

Regarding the specification of service configurations, we have proposed the SYNOPSIS notation. This plain-text language, which takes its inspiration from textual variability languages [53], let describe the decision space of HCSs in a succinct and expressive way while enabling at the same time an automated processing. As far as we know, this is the first notation especially intended to describe the configuration capabilities of services. On the user's side, we have presented a variant of this notation, the so named UCL, to describe user needs on HCSs– in terms of items, requirements and preferences.

A set of validity criteria for SYNOPSIS documents have been proposed in order to detect anomalies in HCSs. We have identified three different categories, depending on the importance: warning, term level and service level. While warning level anomalies only affect the description clarity, term and service level anomalies damages the configuration capabilities of the service. In both cases, dead values – term values which cannot be selected under any circumstance – are the base underlying errors.

Regarding the analysis techniques of HCSs, we have proposed a catalogue of analysis operations. The analysis operations are intended to assist HCS providers and consumers in their decision making. For providers, we propose analysis operations to automatically check the validity criteria of SYNOPSIS specifications, and consequently ensure that the service decision space is free of anomalies. For consumers, we propose analysis operations to assist the search and selection of the best configuration. In both cases, we haven taken inspiration from the AAFM [19] to design the analysis operations for HCSs.

Our automated analysis techniques rely on the formalisation of the underlying SYNOPSIS and UCL models as SFMs [154, 162]. This formalisation enables 1) the use of SFM analysis operations as base for HCS analysis operations and 2) the use of AAFM tools to prototype the HCS-driven solutions.

Regarding the validation of our proposal, we have interpreted two cloud scenarios in terms of HCS specification and automated analysis. For such particular solutions and implementations, we have employed variability modelling and automated analysis techniques to describe and analyse the service decision space. First, we have assisted the migration of an on-premise infrastructure to the Amazon's cloud. And second, we have performed the adaptation analysis of a multi-tenant Desktop as a Service to satisfy the changing preferences of their tenants. The results prove the adequacy and feasibility of modelling and analysis techniques for the decision making in these scenarios.

8.2 Discussion and Future Work

We had two alternative ways to relate this dissertation: focusing on variability or focusing on services. On the one side, we could relate this thesis as an application case of variability modelling and analysis techniques. On the other side, we could relate this thesis defining the problem of HCSs in the services field, and then tackling it by means of the aforementioned techniques. Although both ways are valid, we think that the definition of a problem is, in most of the cases, more valuable – although consequently requires more effort – than an application case.

In a similar way, the formalisation of HCSs could be done in several ways. For instance, we could choose CSP as target domain. Indeed, CSP is a well-known paradigm where we are rather experienced and whose semantic distance with HCSs is short. However, a translation to SFMs present several benefits that other paradigms cannot offer. In particular 1) SFMs are already formalised in CSP and First-order Logic among others, 2) SFMs define a set of analysis operations which are used as a starting point for the HCS operations and 3) the existing AAFM tools can be employed to perform the analysis of SFMs.

A challenging and non-tackled issue in this dissertation is the automated extraction of SYNOPSIS specifications from existing HCSs. This task is performed manually most of the times. Although we could propose a method to perform it, it should be adapted for each specific case, and requires a deep understanding of the HCS. Some tedious tasks can be automated, such as the extraction of the pricing policy and its translation to table constraints by means of web scraping. However, the identification and modelling of the decision terms and their values and dependencies are carried out manually. This issue becomes more challenging when we consider the usual evolution

of HCSs and the necessary maintenance of the model. IaaS providers, for instance, use to update their offers periodically. This is the case of Amazon, which does it every few weeks.

Ontology alignments among similar HCSs from different providers also remains to be tackled. An ontology alignment would enable multi-provider support for the decision making of the consumers. For example, Rackspace servers, Azure virtual machines and Google compute engine are three computing services, as Amazon EC2. However, the employed vocabulary and decision terms slightly change from one vendor to another, e.g. Amazon bills computing instances per hour, while Google bills a minimum of 10 minutes per instance, and after that in spans of 1 minute.

Our proposal also has limitations for the configuration of multiple service items. The number of items required by a consumer has to be specified in order to perform the analysis techniques. This means that the number of instances that a consumer requires is the same number of instances returned by the analysis. The underlying reason is that our analysis approach cannot determine the optimal number of instances to satisfy given user needs. Therefore, in this dissertation we are not able to answer a question like "which is the cheapest combination of instances to satisfy my needs?".

8.3 PUBLICATIONS

Publications Related to this Dissertation

The following publications present the preliminary results that have driven to this dissertation.

International journals

- J. García Galán, P. Trinidad, O.F. Rana, A. Ruiz-Cortés. *Automated Configuration Support for Infrastructure Migration to the Cloud*. Future Generation Computer Systems. 2015. **In Press** [73].

- J. García Galán, L. Pasquale, P. Trinidad, A. Ruiz-Cortés. *User-centric Adaptation Analysis of Multi-tenant Services*. ACM Transactions on Autonomous and Adaptive Systems. 2015 [72]. **Under revision (second round)**.

International conferences, workshops and symposiums

- J. García Galán, O.F. Rana, P. Trinidad, A Ruiz-Cortés. *Migrating to the Cloud: a Software Product Line based analysis*. 3rd International Conference on Cloud Computing and Services Science (CLOSER). 2013 [69]. **Best student paper award**.
- J. García Galán, L. Pasquale, P. Trinidad, A. Ruiz-Cortés. *User-centric adaptation of multi-tenant services: preference-based analysis for service reconfiguration*. 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). Co-located with the International Conference on Software Engineering. 2014 [71]. **Best paper award**.
- J. García Galán, P. Trinidad, A Ruiz-Cortés. *Multi-user variability configuration: A game theoretic approach*. IEEE/ACM International Conference on Automated Software Engineering (ASE). 2013 [70].
- J. García Galán, P. Trinidad, A Ruiz-Cortés. *ISA Packager: a tool for SPL deployment*. Proceedings of the 5th International Workshop on Variability Modelling of Software-intensive Systems, 2011 [67].
- P Trinidad, C Muller, J. García Galán, A Ruiz-Cortés. *Building industry-ready tools: FAMA Framework and ADA*. 3rd International Workshop on Academic Software Development Tools and Techniques (WASDETT). 2010 [159].

National conferences, workshops and symposiums

- J. García Galán, P. Trinidad, R. Capilla. *Automating the deployment of componentized systems*. Jornadas de IngenierÃa del Software y Bases de Datos (JISBD). 2012 [68]. **Best emerging work**.

Other Publications

These publications, although they are not specifically linked to the problems stated in this dissertation, have contributed to increase the knowledge and understanding of the techniques in which the solution of this dissertation relies:

International conferences, workshops and symposiums

- J. García Galán, J.A. Galindo, P Trinidad, A Ruiz-Cortés. *Tool supported error detection and explanations on feature models*. In: Proc. of 2nd International Workshop on Formal Methods and Analysis in Software Product Line Engineering (FMSPLE 2011), co-located with Software Product Line Conference. 2011. [66]

- J.A. Galindo, F. Roos-Frantz, J. García Galán, A Ruiz-Cortés. *Extracting orthogonal variability models from Debian repositories*. In: Proc. of 2nd International Workshop on Formal Methods and Analysis in Software Product Line Engineering (FMSPLE 2011), co-located with Software Product Line Conference. 2011. [61]

National conferences, workshops and symposiums

- Carlos Müller, Jesús García-Galán, Antonio Ruiz-Cortés, M. Resinas. *ADA: Agreement Documents Analyser*. JSWEB 2010 Tool demo. Valencia; 2010. [117]

- P. Trinidad, J. García Galán, A Ruiz-Cortés. *FaMa Abductive: una herramienta para explicaciones de errores en modelos de caracterÃsticas*. Actas de las XVI Jornadas de Ingeniería del Software y Bases de Datos. A Coruña; 2011. [160]

- J.A. Galindo, F. Roos-Frantz, A Ruiz-Cortés, J. García Galán. *Automated Analysis of Diverse Variability Models with Tool Support*. Actas de las XIX Jornadas de Ingeniería del Software y Bases de Datos. Cádiz; 2014 [136]

BIBLIOGRAPHY

[1] M. Acher, P. Collet, P. Lahire, and R. B. France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming*, 78(6):657–681, 2013. (pages 27, 28, 34).

[2] R. Ali, C. Solis, I. Omoronyia, M. Salehie, and B. Nuseibeh. Social Adaptation: When Software Gives Users a Voice. In *Proc. of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2012. (pages 120, 128, 148).

[3] Amazon. AWS Total Cost of Ownership (TCO) Calculator. https://awstcocalculator.com, 2014. (pages 6, 40, 90, 110).

[4] Amazon. Amazon Web Services. http://aws.amazon.com/, 2015. (pages 5, 18).

[5] Amazon. Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2, 2015. (pages 4, 17, 18, 19, 20, 90, 96, 99, 100).

[6] Amazon. Amazon Sample Storage Service. http://aws.amazon.com/s3/, 2015. (page 18).

[7] Amazon Web Services. Amazon EC2 calculator. http://calculator.s3.amazonaws.com/calc5.html. Last accessed: November 2012, 2014. (pages 6, 40).

[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and Others. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010. (pages 4, 16).

[9] M. Asadi, S. Soltani, D. Gasevic, M. Hatala, and E. Bagheri. Toward automated feature model configuration with optimizing non-functional requirements. *Information and Software Technology*, 56(9):1144–1165, 2014. (pages 25, 34, 137).

[10] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. ISBN 0-521-78176-0. (page 46).

[11] R. Bachmeyer and H. Delugach. A conceptual graph approach to feature modeling. In U. Priss, S. Polovina, and R. Hill, editors, *Conceptual Structures: Knowledge Architectures for Smart Applications*, volume 4604 of *Lecture Notes in Computer Science*, pages 179–191. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-73680-6. (page 46).

[12] K. Bak, K. Czarnecki, and A. Wasowski. Feature and meta-models in clafer: mixed, specialized, and coupled. In *Software Language Engineering*, pages 102–122. Springer, 2011. (pages 27, 28).

[13] L. Baresi, S. Guinea, and L. Pasquale. Service-Oriented Dynamic Software Product Lines. *IEEE Computer*, 2012. (pages 120, 130).

[14] A. Barros and D. Oberle. *Handbook of Service Description: USDL and Its Methods*. Springer Publishing Company, Incorporated, 2012. (page 5).

[15] D. Batory. Feature models, grammars, and propositional formulas. In *Software Product Lines Conference, LNCS 3714*, pages 7–20, 2005. (pages 26, 28, 46).

[16] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and P. F. Patel-Schneider. *OWL Web Ontology Language Reference*. World Wide Web Consortium, 2004. URL http://www.w3.org/TR/owl-ref/. (page 46).

[17] D. Benavides. *On the Automated Analysis of Software Product Lines Using Feature Models. A framework for developing automated tool support.* PhD thesis, University of Seville, http://www.lsi.us.es/~dbc/dbc_archivos/pubs/benavides07-phd.pdf, 2007. (pages 7, 156).

[18] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005. ISSN 0302-9743. (pages 21, 24, 29, 42, 46, 93, 107).

[19] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 9 2010. (pages 5, 29, 42, 44, 45, 93, 102, 105, 107, 121, 137, 156).

[20] N. Bencomo, P. Sawyer, G. S. Blair, and P. Grace. Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems. In *Software Product Line Conference*, 2008. (page 150).

[21] T. Berger, A. Wasowski, K. Czarnecki, S. She, and R. Lotufo. A Study of Variability Models and Languages in the Systems Software Domain. *IEEE Transactions on Software Engineering*, 2013. (page 5).

[22] J. Bernal Bernabe, J. M. Marin Perez, J. M. Alcaraz Calero, F. J. Garcia Clemente, G. Martinez Perez, and A. F. Gomez Skarmeta. Semantic-aware multi-tenancy authorization system for cloud architectures. *Future Generation Computer Systems*, 32:154–167, 2014. (page 150).

[23] D. L. Berre. Sat4j: The satisfiability library for java (www.sat4j.org). http://www.sat4j.org, 2004. (page 46).

[24] P. V. Beserra, A. Camara, R. Ximenes, A. B. Albuquerque, and N. C. Mendonca. Cloudstep: A step-by-step decision process to support legacy application migration to the cloud. In *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012 IEEE 6th International Workshop on the*, pages 7–16. IEEE, 2012. (pages 94, 114).

[25] C.-P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. t Hart. Enabling multi-tenancy: An industrial experience report. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–8. IEEE, 2010. (pages 120, 122).

[26] D. Borgetto, M. Maurer, G. Da-Costa, J.-M. Pierson, and I. Brandic. Energy-efficient and SLA-aware management of IaaS clouds. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, page 10, 2012. (page 115).

[27] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009. (page 4).

[28] C. Cabanillas. *Enhancing the Management of Resource-Aware Business Processes*. PhD thesis, Universidad de Sevilla, 12/2012 2012. (pages 7, 156).

[29] C. Cabanillas, J. M. Garc'ia, M. Resinas, D. Ruiz, J. Mendling, and A. Ruiz-Cortés. Priority-Based Human Resource Allocation in Business Processes. In *ICSOC*, 2013. (page 136).

[30] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. Lo Presti, and R. Mirandola. Moses: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems. *IEEE Transactions on Software Engineering*, 38(5):1138–1159, 2012. (pages 42, 120, 147, 148).

[31] S. Caton and O. Rana. Towards autonomic management for Cloud services based upon volunteered resources. *Concurrency and Computation: Practice and Experience*, 2012. (pages 42, 149).

[32] E. Cavalcante, A. Almeida, and T. Batista. Exploiting software product lines to develop cloud computing applications. In *Software Product Line Conference*, pages 179–187, 2012. (page 115).

[33] Citrix. Citrix Virtual Desktop Handbook 7.x. http://support.citrix.com/article/CTX139331, Last 2013. (pages xv, 17, 122, 123, 124, 125).

[34] D. Clarke, R. Muschevici, J. Proença, I. Schaefer, and R. Schlatte. Variability modelling in the abs language. In *Formal Methods for Components and Objects*, pages 204–224. Springer, 2012. (pages 27, 28).

[35] A. Classen, Q. Boucher, and P. Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Science of Computer Programming*, 76(12): 1130–1143, 2011. (pages 27, 28).

[36] CloudHarmony. Cloud Harmony. http://cloudharmony.com/, 2014. (pages 90, 116).

[37] Cloudorado. Cloudorado. http://www.cloudorado.com/, 2014. (pages 41, 116).

[38] CloudScreener. Cloud Screener. http://www.cloudscreener.com/, 2014. (pages 6, 40, 90, 111, 116).

[39] M. Cordy, P.-Y. Schobbens, P. Heymans, and A. Legay. Beyond boolean product-line model checking: dealing with feature attributes and multi-features. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 472–481. IEEE Press, 2013. (pages 26, 27, 28).

[40] K. Czarnecki and P. Kim. Cardinality-based feature modeling and constraints: A progress report. In *Proceedings of the International Workshop on Software Factories At OOPSLA 2005*, 2005. (page 46).

[41] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In *Proceedings of the Third Software Product Line Conference 2004*, pages 266–282. Springer, LNCS 3154, 2004. (pages 32, 34).

[42] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005. doi: 10.1002/spip.213. URL http://dx.doi.org/10.1002/spip.213. (pages 22, 28, 31).

[43] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, Apr. 2005. doi: 10.1002/spip.225. URL http://dx.doi.org/10.1002/spip.225. (page 26).

[44] F. Dalpiaz, E. Serral, P. Valderas, P. Giorgini, and V. Pelechano. A NFR-based framework for user-centered adaptation. In *Conceptual Modeling*. Springer, 2012. (page 148).

[45] A. V. Dastjerdi, S. G. Tabatabaei, and R. Buyya. An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010. (page 116).

[46] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2): 182–197, 2002. (page 141).

[47] A. del Río-Ortega. *On the Definition and Analysis of Process Performance Indicators*. International, University of Sevilla, 09/2012 2012. (pages 7, 156).

[48] A. v. Deursen and P. Klint. Domain–specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002. (pages 26, 28).

[49] B. Dougherty, J. White, and D. C. Schmidt. Model-driven auto-scaling of green cloud computing infrastructure. *Future Generation Computer Systems*, 2012. (page 115).

[50] I. Dropbox. Dropbox. https://www.dropbox.com/, 2014. (pages 4, 5, 17, 19, 20).

[51] J. J. Durillo and A. J. Nebro. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 2011. (page 141).

[52] E. Ostrom. *Governing the Commons*. CUP, 1990. (page 149).

[53] H. Eichelberger and K. Schmid. A systematic analysis of textual variability modeling languages. In *Proceedings of the 17th International Software Product Line Conference*, pages 12–21. ACM, 2013. (pages 26, 28, 109, 156).

[54] H. Eskandari, C. D. Geiger, and G. B. Lamont. Fastpga: A dynamic population sizing approach for solving expensive multiobjective optimization problems. In *Evolutionary Multi-Criterion Optimization*, pages 141–155. Springer, 2007. (page 141).

[55] F. Fittkau, S. Frey, and W. Hasselbring. Cdosim: Simulating cloud deployment options for software migration support. In *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012 IEEE 6th International Workshop on the*, pages 37–46. IEEE, 2012. (pages 6, 114).

[56] P. Flach. *Simply Logical*. John Wiley, April 1994. ISBN 0-471-94152-2. (page 46).

[57] S. Frey and W. Hasselbring. The cloudmig approach: Model-based migration of software systems to cloud-optimized applications. *International Journal on Advances in Software*, 4(3 and 4):342–353, 2011. (page 114).

[58] S. Frey, F. Fittkau, and W. Hasselbring. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 512–521. IEEE Press, 2013. (pages 6, 90, 94, 114).

[59] K. Z. Gajos, M. Czerwinski, D. S. Tan, and D. S. Weld. Exploring the Design Space for Adaptive Graphical User Interfaces. In *Proc. of the International Working Conference on Advanced Visual Interfaces*, pages 201–208. ACM Press, 2006. (page 129).

[60] J. A. Galindo. *Evolution, testing and configuration of variability intensive systems*. PhD thesis, University of Seville/University of Rennes 1, 3 2015. Advised by David Benavides and Benoit Baudry. (pages 7, 156).

[61] J. A. Galindo, F. Roos-Frantz, J. García-Galán, and A. Ruiz-Cortés. Extracting orthogonal variability models from debian repositories. In *15th International Software Product Line Conference Proceedings*, volume 2, 2011. (page 160).

[62] S. Gallacher, E. Papadopoulou, N. K. Taylor, and M. H. Williams. Learning user preferences for adaptive pervasive environments: an incremental and temporal approach. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 8(1):5, 2013. (pages 148, 152).

[63] J. M. García, D. Ruiz, and A. Ruiz-Cortés. A model of user preferences for semantic services discovery and ranking. In *The Semantic Web: Research and Applications*, pages 1–14. Springer, 2010. (page 118).

[64] J. M. García, D. Ruiz, and A. Ruiz-Cortés. An Intuitive and Formal Description of Preferences for Semantic Web Service Discovery and Ranking. Technical report, University of Seville, 2012. (page 37).

[65] J. M. García, M. Junghans, D. Ruiz, S. Agarwal, and A. Ruiz-Cortés. Integrating Semantic Web Services Ranking Mechanisms Using a Common Preference Model. *Knowledge-Based Systems*, 2013. (pages 36, 68, 121, 131, 136).

[66] J. García-Galán, P. Trinidad, J. Á.Galindo, and A. Ruiz-Cortés. Tool supported error detection and explanations on feature models. In *Proc. of 2nd International Workshop on Formal Methods and Analysis in Software Product Line Engineering (FM-SPLE 2011), co-located with Software Product Line Conference 2011 (SPLC 2011)*, Munich, 2011. Fraunhofer, Fraunhofer. (page 160).

[67] J. García-Galán, P. Trinidad, and A. Ruiz-Cortés. Isa packager: A tool for spl deployment. In *Proceedings of the Fifth International Workshop on Variability Modelling of Software–intensive Systems (VAMOS)*, 2011. (page 159).

[68] J. García-Galán, P. Trinidad, and R. Capilla. Automating the deployment of componentized systems. In *Actas de las XVIII Jornadas de Ingeniería del Software y Bases de Datos*, Almeria, Sept. 2012. (page 159).

[69] J. García-Galán, O. F. Rana, P. Trinidad, and A. Ruiz-Cortés. Migrating to the Cloud: a Software Product Line based analysis. In *3rd International Conference on Cloud Computing and Services Science (CLOSER)*, pages 416–426, 2013. (pages 5, 29, 90, 92, 131, 132, 151, 159).

[70] J. García-Galán, P. Trinidad, and A. Ruiz-Cortés. Multi-user Variability Configuration: A Game Theoretic Approach. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013. (pages 151, 159).

[71] J. García-Galán, L. Pasquale, P. Trinidad, and A. R. Cortés. User-centric adaptation of multi-tenant services: preference-based analysis for service reconfiguration. In *SEAMS*, pages 65–74, 2014. (pages 117, 120, 159).

[72] J. García-Galán, L. Pasquale, P. Trinidad, and A. R. Cortés. User-centric Adaptation Analysis of Multi-tenant Services. *Transactions on Autonomous and Adaptive Systems*, 2014. Submitted. (page 158).

[73] J. García-Galán, P. Trinidad, O. F. Rana, and A. Ruiz-Cortés. Automated Configuration Support for Infrastructure Migration to the Cloud. *Future Generation Computer Systems*, 2015. In Press. (pages 4, 6, 41, 158).

[74] Google. Google App Engine. https://appengine.google.com/, 2015. (page 17).

[75] Google Inc. Google Compute engine. https://cloud.google.com/compute/, 2014. (page 96).

[76] J. Guo, J. White, G. Wang, J. Li, and Y. Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *J. Syst. Softw.*, 2011. (pages 121, 141, 151).

[77] V. Haarslev and R. Moller. Description of the RACER system and its applications. In *Description Logics*, 2001. URL `http://www.racer-systems.com`. (page 46).

[78] T. Han and K. M. Sim. An ontology-enhanced cloud service discovery system. In *Proc. of the International MultiConference of Engineers and Computer Scientists, Hong Kong*, 2010. (page 116).

[79] A. Hemakumar. Finding contradictions in feature models. In *First International Workshop on Analyses of Software Product Lines (ASPL'08)*, pages 183–190, 2008. (page 46).

[80] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le Traon. Towards automated testing and fixing of re-engineered feature models. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 1245–1248. IEEE, 2013. (page 45).

[81] I. Heroku. Heroku. https://www.heroku.com/, 2015. (page 17).

[82] A. H. M. T. Hofstede and H. Proper. How to formalize it? formalization principles for information system development methods. *Information and Software Technology*, 40:519–540, 1998. (pages 7, 72).

[83] C. Inzinger, B. Satzger, P. Leitner, W. Hummer, and S. Dustdar. Model-based Adaptation of Cloud Computing Applications. In *International Conference on Model-Driven Engineering and Software Development*, 2013. (page 149).

[84] ISA Research group. Fama tool suite. http://www.isa.us.es/fama/, 2014. (pages 28, 109).

[85] ISO. Iso/iec 13211-1. international standard, information technology - programming languages - prolog - part 1: General core, 1995. (page 46).

[86] P. Jamshidi, A. Ahmad, and C. Pahl. Cloud migration research: A systematic review. *Cloud Computing, IEEE Transactions on*, 1(2):142–157, July 2013. ISSN 2168-7161. doi: 10.1109/TCC.2013.10. (pages 6, 41, 94, 113).

[87] W. Ju and L. Leifer. The Design of Implicit Interactions: Making Interactive Systems Less Obnoxious. *Design Issues*, 24(3):72–84, 2008. (page 135).

[88] K. C. Kang. Feature-oriented domain analysis (FODA) feasibility study. Technical Report November, Software Engineering Institute, 1990. (pages 5, 21, 22, 28, 30, 46, 92, 97, 121, 124).

[89] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36 (1):41–50, 2003. doi: 10.1109/MC.2003.1160055. URL http://dx.doi.org/10.1109/MC.2003.1160055. (page 120).

[90] A. Khajeh-Hosseini, I. Sommerville, J. Bogaerts, and P. B. Teregowda. Decision Support Tools for Cloud Migration in the Enterprise. In *IEEE CLOUD Conference*, pages 541–548, 2011. (pages 6, 114).

[91] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, and I. Sommerville. The Cloud Adoption Toolkit: supporting cloud adoption decisions in the enterprise. *Softw., Pract. Exper.*, pages 447–465, 2012. (page 114).

[92] I. Kumara, J. Han, A. Colman, T. Nguyen, and M. Kapuruge. Sharing with a Difference: Realizing Service-Based SaaS Applications with Runtime Sharing and Variation in Dynamic Software Product Lines. In *10th International Conference on Services Computing*, 2013. (page 120).

[93] Y.-W. Kwon and E. Tilevich. Cloud refactoring: automated transitioning to cloud-based services. *Automated Software Engineering*, 21(3):345–372, 2014. (page 114).

[94] S. Lamparter, A. Ankolekar, R. Studer, and S. Grimm. Preference-based selection of highly configurable web services. In *Proceedings of the 16th international conference on World Wide Web*, pages 1013–1022. ACM, 2007. (pages 16, 21, 148).

[95] S. Le, H. Dong, F. K. Hussain, O. K. Hussain, and E. Chang. Cloud service selection: State-of-the-art and future research directions. *Journal of Network and Computer Applications*, 2014. (pages 6, 41).

[96] K. Lee, K. C. Kang, and J. Lee. Concepts and guidelines of feature modeling for product line software engineering. In *Software Reuse: Methods, Techniques, and Tools*, pages 62–77. Springer, 2002. (page 96).

[97] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas. Migration of multi-tier applications to infrastructure-as-a-service clouds: An investigation using kernel-based virtual machines. In *Grid Computing (GRID), 2011 12th IEEE/ACM International Conference on*, pages 137–144. IEEE, 2011. (pages 6, 115).

[98] S. Malek, N. Medvidovic, and M. Mikic-Rakic. An extensible framework for improving a distributed software system's deployment architecture. *IEEE Transactions on Software Engineering*, 38(1):73–100, 2012. (pages 120, 147, 148).

[99] R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 2004. (page 121).

[100] C. C. Marquezan, F. Wessling, A. Metzger, K. Pohl, C. Woods, and K. Wallbom. 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*, pages 48–57. ACM, 2014. (page 150).

[101] O. Martín-Díaz. *Emparejamiento Automático de Servicios Web usando Programación con Restricciones*. phd, Dpto. de Lenguajes y Sistemas Informáticos, E.T.S. de Ingeniería Informática. Universidad de Sevilla, 2007. (pages 7, 156).

[102] O. Martín-Díaz, A. Ruiz-Cortés, A. Durán, and C. Müller. An approach to temporal-aware procurement of web services. In *Service-Oriented Computing-ICSOC 2005*, pages 170–184. Springer, 2005. (page 118).

[103] M. Maurer, I. Brandic, and R. Sakellariou. Adaptive resource configuration for Cloud infrastructure management. *Future Generation Computer Systems*, 2013. doi: 10.1016/j.future.2012.07.004. URL http://dx.doi.org/10.

1016/j.future.2012.07.004. ¡ce:title¿Special section: Recent advances in e-Science¡/ce:title¿. (pages 42, 140, 149).

[104] P. Mell and T. Grance. The NIST definition of cloud computing. *NIST special publication*, 800(145):7, 2011. (pages 4, 16).

[105] M. Mendonca, M. Branco, and D. Cowan. SPLOT: software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*. ACM, 2009. (pages 27, 28).

[106] M. Mendonça, T. T. Bartolomei, and D. Cowan. Decision-Making Coordination in Collaborative Product Configuration. In *23rd Annual ACM Symposium on Applied Computing, SAC*, 2008. (page 31).

[107] M. Mendonça, D. Cowan, W. Malyk, and T. Oliveira. Collaborative Product Configuration : Formalization and Efficient Algorithms for Dependency Analysis. *Journal of Software*, 2008. (pages 32, 34).

[108] M. Mendonça, A. Wasowski, and K. Czarnecki. Sat-based analysis of feature models is easy. In *SPLC*, pages 231–240, 2009. (page 46).

[109] M. Menzel and R. Ranjan. Cloudgenius: decision support for web server cloud migration. In *Proceedings of the 21st international conference on World Wide Web*, pages 979–988. ACM, 2012. (pages 6, 114).

[110] R. Michel, A. Classen, A. Hubaux, and Q. Boucher. A formal semantics for feature cardinalities in feature diagrams. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, pages 82–89. ACM, 2011. (page 26).

[111] Microsoft. Windows Azure Configurator. http://azure.microsoft.com/en-us/pricing/calculator/, 2014. (page 40).

[112] Microsoft. Azure Virtual Machines. http://azure.microsoft.com/en-us/services/virtual-machines/, 2014. (pages 19, 96).

[113] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl. Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications. In *ICSE Workshop on Principles of Engineering Service Oriented Systems*, 2009. (pages 120, 150).

[114] P. Mohagheghi and T. Sæther. Software engineering challenges for migration to the service cloud paradigm: Ongoing work in the remics project. In *Services (SERVICES), 2011 IEEE World Congress on*, pages 507–514. IEEE, 2011. (page 115).

[115] C. Müller. *On the Automated Analysis of WS-Agreement Documents. Applications to the Processes of Creating and Monitoring Agreements*. International dissertation, Universidad de Sevilla, 2013. (pages 5, 7, 156).

[116] C. Müller, O. Martín-Díaz, A. Ruiz-Cortés, M. Resinas, and P. Fernandez. *Improving temporal-awareness of WS-agreement*. Springer, 2007. (page 118).

[117] C. Müller, J. García-Galán, A. Ruiz-Cortés, and M. Resinas. Ada: Agreement documents analyser*. In *JSWEB 2010*, Valencia, Sep 2010. (page 160).

[118] R. B. Myerson. *Game theory: analysis of conflict*. Harvard University Press, 1991. (page 138).

[119] V. Nallur and R. Bahsoon. A decentralized self-adaptation mechanism for service-based applications in the cloud. *Software Engineering, IEEE Transactions on*, 39(5):591–612, 2013. (pages 42, 150).

[120] Y. V. Natis. Gartner Reference Model for Elasticity and Multitenancy. Technical report, Gartner, Inc., 2012. (pages 120, 122).

[121] L. D. Ngan and R. Kanagasabai. Owl-s based semantic cloud service broker. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 560–567. IEEE, 2012. (page 116).

[122] A. Nöhrer and A. Egyed. Conflict resolution strategies during product configuration. In D. Benavides, D. S. Batory, and P. Grünbacher, editors, *VaMoS*, volume 37 of *ICB-Research Report*, pages 107–114. Universität Duisburg-Essen, 2010. (page 34).

[123] T. Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Programmers. Pragmatic Bookshelf, May 2007. ISBN 0978739256.

[124] R. B. Penman, T. Baldwin, and D. Martinez. Web scraping made simple with sitescraper, 2009. (page 108).

[125] J. Pitt, J. Schaumeier, D. Busquets, and S. Macbeth. Self-organising common-pool resource allocation and canons of distributive justice. In *Self-Adaptive and*

Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on, pages 119–128. IEEE, 2012. (page 149).

[126] PlanForCloud. Plan for cloud. http://www.planforcloud.com/, 2014. (page 116).

[127] I. project consortium. Open variability modeling approach for service ecosystems. technical report deliverable. Technical report, INDENICA research project, 2012. (pages 27, 28).

[128] C. Quinton, D. Romero, and L. Duchien. Automated selection and configuration of cloud environments using software product lines principles. In *IEEE CLOUD 2014*, page 8, 2014. (pages 29, 30, 94, 115).

[129] Rackspace. Rackspace. http://www.rackspace.com/, 2014. (page 5).

[130] Rackspace. Rackspace Solutions Configurator. http://www.rackspace.co.uk/solutions-configurator, 2014. (pages 6, 40, 90).

[131] Rackspace. Rackspace Servers. http://www.rackspace.com/cloud/servers, 2015. (pages 17, 96).

[132] M.-O. Reiser. *Core Concepts of the Compositional Variability Management Framework (CVM): A Practitioner's Guide*. TU, Professoren der Fak. IV, 2009. (pages 26, 28, 34).

[133] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Extending feature diagrams with uml multiplicities. In *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, June 2002. (pages 22, 26).

[134] M. Á. Rodríguez-García, R. Valencia-García, F. García-Sánchez, and J. J. Samper-Zapater. Ontology-based annotation and retrieval of services in the cloud. *Knowledge-Based Systems*, 56:15–25, 2014. (page 116).

[135] F. Roos-Frantz, D. Benavides, A. Ruiz-Cortés, A. Heuer, and K. Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal*, pages 1–47, 2012. ISSN 0963-9314. (page 34).

[136] F. Roos-Frantz, J. A. G. Duarte, D. Benavides, A. R. Cortés, and J. García-Galán. Automated analysis of diverse variability models with tool support. In *Jornadas*

de Ingeniería del Software y de Bases de Datos (JISBD 2014), Cádiz. Spain, 2014. (page 160).

[137] M. Rosenmüller, N. Siegmund, T. Thüm, and G. Saake. Multi-dimensional variability modeling. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '11*, 2011. (pages 27, 28, 34).

[138] A. Ruiz-Cortés. *Una Aproximación Semicualitativa al Tratamiento Automático de Requisitos de Calidad Aplicación a la Obtención Automática de Acuerdos de Nivel de Servicio en MOWS*. Phd thesis, Universidad de Sevilla, 2002. (pages 7, 156).

[139] A. B. Sánchez, S. Segura, and A. Ruiz-Cortés. The Drupal Framework: A Case Study to Evaluate Variability Testing Techniques. In *Proc. of the 8th Int. Work. on Variability Modelling of Software-intensive Systems*, page 11, 2014. (page 132).

[140] A. S. Sayyad, J. Ingram, Tim, and H. Ammar. Scalable Product Line Configuration: A Straw to Break the Camel's Back. In *International Conference on Automated Software Engineering (ASE)*, 2013. (pages 121, 141, 142, 144, 151).

[141] A. S. Sayyad, T. Menzies, and H. Ammar. On the value of user preferences in search-based software engineering: a case study in software product lines. In *International Conference on Software Engineering*, 2013. (page 25).

[142] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis, Minnesota, USA, Sept. 2006. (pages 22, 23, 35).

[143] J. Schroeter, S. Cech, S. Götz, C. Wilke, and U. Assmann. Towards Modeling a Variable Architecture for Multi-tenant SaaS-Applications. In *International Workshop on Variability Modelling of Software-Intensive Systems*. ACM, 2012. (pages 120, 150).

[144] J. Schroeter, P. Mucha, M. Muth, K. Jugel, and M. Lochau. Dynamic configuration management of cloud-based applications. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*, pages 171–178, 2012. (pages 30, 115, 120, 130, 150).

[145] S. Segura, J. Á.Galindo, D. Benavides, J. A. Parejo, and A. Ruiz-Cortés. BeTTy: Benchmarking and Testing on the Automated Analysis of Feature Models. In *VaMoS*, Leipzig, Germany, 2012. ACM, ACM. (page 144).

[146] S. Segura, A. B. Sánchez, and A. Ruiz-Cortés. Automated variability analysis and testing of an e-commerce site.: an experience report. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 139–150. ACM, 2014. (page 25).

[147] H. Song, S. Barrett, A. Clarke, and S. Clarke. Self-adaptation with End-User Preferences: Using Run-Time Models and Constraint Solving. In *Model-Driven Engineering Languages and Systems*. Springer, 2013. (page 148).

[148] C. Speier, I. Vessey, and J. S. Valacich. The Effects of Interruptions, Task Complexity, and Information Presentation on Computer-Supported Decision-Making Performance. *Decision Sciences*, 34(4):771–797, 2003. (page 128).

[149] J. Stein, I. Nunes, and E. Cirilo. Preference-based Feature Model Configuration with Multiple Stakeholders. In *Proc. of the 18th Int. Software Product Lines Conf.*, pages 132–141, 2014. (page 151).

[150] S. Sundareswaran, A. Squicciarini, and D. Lin. A Brokerage-Based Approach for Cloud Service Selection. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 558–565, 2012. (pages 94, 115).

[151] B. C. Tak, B. Urgaonkar, and A. Sivasubramaniam. To move or not to move: The economics of cloud computing. In *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, pages 5–5. USENIX Association, 2011. (pages 90, 94, 115).

[152] K. Takemura. *Behavioral Decision Theory*. Springer, 2014.

[153] T. Thum, C. Kastner, S. Erdweg, and N. Siegmund. Abstract Features in Feature Modeling. In *Proceedings of the 2011 15th International Software Product Line Conference*, page 10, 2011. (pages 92, 124).

[154] P. Trinidad. *Automating the Analysis of Stateful Feature Models*. PhD thesis, University of Seville,
urlhttp://www.lsi.us.es/~trinidad/docs/tesis.pdf, 2012. (pages 7, 9, 28, 29, 34, 35, 45, 156).

[155] P. Trinidad and A. Ruiz-Cortés. Abductive reasoning and automated analysis of feature models: How are they connected? In *3rd. International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*, pages 145–153, Sevilla,

Spain, Jan 2009. ICB Research Report N. 29. URL http://www.vamos-workshop.net/. (page 44).

[156] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and M. Toro. Explanations for agile feature models. In *Procceedings of the 1st International Workshop on Agile Product Line Engineering (APLE'06)*, 2006. (page 46).

[157] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, and M. Toro. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, 81(6):883–896, 2008. doi: 10.1016/j.jss.2007.10.030. (pages 46, 103, 104).

[158] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez. FAMA Framework. In *12th Software Product Lines Conference (SPLC)*, pages 359–359, 2008. (pages 8, 9, 28, 46, 106, 109, 133, 142).

[159] P. Trinidad, C. Müller, J. García-Galán, and A. Ruiz-Cortés. Building industry-ready tools: Fama framework and ada. In *Third International Workshop on Academic Software Development Tools and Techniques*, pages 160–173, 2010. (page 159).

[160] P. Trinidad, J. García-Galán, and A. Ruiz-Cortés. FaMa Abductive: una herramienta para explicaciones de errores en modelos de caracteristicas. In *XVI Jornadas de Ingeniería del Software y Bases de Datos*, 2011. (page 160).

[161] P. Trinidad, A. Ruiz-Cortés, and D. Benavides. Automated analysis of stateful feature models. In *Seminal Contributions to Information Systems Engineering*, pages 375–380. Springer, 2013. (page 45).

[162] P. Trinidad, A. Ruiz-Cortés, and J. G. Galán. Configurable feature models. In *Actas de las XIX Jornadas de Ingeniería del Software y Bases de Datos*, pages 335–348, 09/2014 2014. ISBN 84-697-1152-0. (pages 9, 28, 29, 34, 156).

[163] I. Trummer, F. Leymann, R. Mietzner, and W. Binder. Cost-optimal outsourcing of applications into the clouds. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 135–142. IEEE, 2010. (pages 94, 114).

[164] H.-L. Truong and S. Dustdar. Composable cost estimation and monitoring for computational applications in cloud computing environments. *Procedia Computer Science*, 1(1):2175–2184, 2010. (page 114).

[165] W.-T. Tsai, G. Qi, and Y. Chen. A Cost-Effective Intelligent Configuration Model in Cloud Computing. In *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, pages 400–408, 2012. (pages 94, 114).

[166] B. Vankeirsbilck, L. Deboosere, P. Simoens, P. Demeester, F. De Turck, and B. Dhoedt. User Subscription-Based Resource Management for Desktop-as-a-Service Platforms. *The Journal of Supercomputing*, 69(1):412–428, 2014. (page 149).

[167] S. Venticinque, R. Aversa, B. Di Martino, and D. Petcu. Agent based cloud provisioning and management: Design and prototypal implementation. In *CLOSER 2011 - Proceedings of the 1st International Conference on Cloud Computing and Services Science*, pages 184–191, 2011. (page 115).

[168] T. von der Massen and H. Lichter. Requiline: A requirements engineering tool for software product lines. In F. van der Linden, editor, *Proceedings of the Fifth International Workshop on Product Family Engineering (PFE-5)*, LNCS 3014, Siena, Italy, 2003. Springer Verlag. (page 46).

[169] T. von der Massen and H. Lichter. Deficiencies in feature models. In T. Mannisto and J. Bosch, editors, *Workshop on Software Variability Management for Product Derivation - Towards Tool Support*, 2004. (page 46).

[170] H. Wang, Y. F. Li, J. Sun, and H. A. Zhang. A Semantic Web Approach to Feature Modeling and Verification. In *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, 2005. (page 46).

[171] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong. A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, 54(2):252–269, 2010. (page 149).

[172] J. Whaley. Javabdd. http://javabdd.sourceforge.net/, 2007. (page 46).

[173] J. White, B. Dougherty, and D. C. Schmidt. Selecting highly optimal architectural feature sets with Filtered Cartesian Flattening. *Journal of Systems and Software*, 2009. doi: 10.1016/j.jss.2009.02.011. URL http://dx.doi.org/10.1016/j.jss.2009.02.011. (page 34).

[174] J. White, D. Benavides, D. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes. Automated diagnosis of feature model configurations. *Journal of Systems and Software*, 83(7):1094 – 1107, 2010. ISSN 0164-1212. doi: DOI: 10.1016/j.jss.2010.02.017. (pages 31, 34, 46, 104, 151).

[175] J. White, B. Dougherty, C. Thompson, and D. C. Schmidt. ScatterD : Spatial Deployment Optimization with Hybrid Heuristic / Evolutionary Algorithms. *ACM Transactions on Autonomous and Adaptive Systems*, 6(3), 2011. doi: 10.1145/2019583.2019585. URL http://dx.doi.org/10.1145/2019583.2019585. (page 45).

[176] E. Wittern and C. Zirpins. Service feature modeling: modeling and participatory ranking of service design alternatives. *Software & Systems Modeling*, pages 1–26, 2014. (pages 29, 30, 115, 116).

[177] E. Wittern, J. Kuhlenkamp, and M. Menzel. Cloud service selection based on variability modeling. In *Service-Oriented Computing*, pages 127–141. Springer, 2012. (pages 5, 30, 115).

[178] Wordpress. Wordpress.com. http://www.wordpress.com/, 2015. (pages 17, 42).

[179] S. Zardari and R. Bahsoon. Cloud adoption: a goal-oriented requirements engineering approach. In *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing*, pages 29–35. ACM, 2011. (page 115).

[180] G. Zhang, H. Ye, and Y. Lin. Quality attribute modeling and quality aware product configuration in software product lines. *Software Quality Journal*, pages 1–37, 2013. (page 34).

[181] W. Zhang, H. Zhao, and H. Mei. A propositional logic-based method for verification of feature models. In J. Davies, editor, *ICFEM 2004*, volume 3308, pages 115–130. Springer–Verlag, 2004. doi: 10.1007/b102837. (page 46).

[182] W. Zhang, H. Mei, and H. Zhao. Feature-driven requirement dependency analysis and high-level software design. *Requirements Engineering*, 11(3):205–220, June 2006. doi: 10.1007/s00766-006-0033-x. (page 46).

[183] W. Zhang, H. Yan, H. Zhao, and Z. Jin. A bdd-based approach to verifying clone-enabled feature models' constraints and customization. In H. Mei, editor, *High Confidence Software Reuse in Large Systems*, volume 5030 of *Lecture Notes in Computer Science*, pages 186–199. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-68062-8. (page 46).