Future Generation Computer Systems 95 (2019) 52–67

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Support mechanisms for cloud configuration using XML filtering techniques: A case study in SaaS

Yang Cao, Chung-Horng Lung *, Samuel A. Ajila, Xiaolin Li

Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada K1S 5B6

HIGHLIGHTS

- Multi-tenancy can considerably complicate SaaS management and deployment.
- Formal methods used for multi-tenancy configuration management is complicated.
- XML filtering methods have been used for information dissemination.
- Yfilter XML filtering method to support SaaS configuration management is described.

ARTICLE INFO

Article history:
Received 13 August 2018
Received in revised form 26 October 2018
Accepted 12 December 2018
Available online 2 January 2019

Keywords:
Cloud Computing
Software-as-a-Service
Multi-Tenancy
Feature Modeling
XML Filtering
Yfilter

ABSTRACT

Software-as-a-service (SaaS) has attracted substantial attention as a software delivery and service model in a cloud computing environment. SaaS delivery can help organizations significantly reduce the cost of using software, because the resources for running SaaS applications are shared among tenants (end users or organizations). However, multi-tenancy can considerably complicate SaaS development, deployment, and maintenance as a result of a large number of co-existing tenant-specific constraints or features. Manually configuring and maintaining tenant-specific features will increase the cost, introduce possible errors, and limit scalability and flexibility. The paper addresses the problem of large variations and complex configurations. Specifically, the objective is to develop mechanisms to support automatic multi-tenant software features analysis and matching for the purposes of efficient deployment and operations in the cloud. The emphasis of this paper is on the matching between the tenant-specific requirements and the SaaS features managed by the cloud provider. This paper proposes a novel approach for cloud feature matching using XML filtering techniques to support the process of multi-tenant SaaS deployment and management. Feature modeling has been widely used to capture requirements and constraints. On the other hand, XML filtering techniques are mature and have been adopted in various problem domains. We used Yfilter, a proven and XML filtering technique, to support two multi-tenant applications: (i) Identifying SaaS configurations (in XPath representations) that satisfy tenant-specific requirements and constraints (in XML notation); and (ii) Identifying tenants that have subscribed to a specific set of SaaS features. The applications can effectively facilitate SaaS subsequent management and operations due to various changes, e.g., functionalities, constraints, cost, etc. The experimental results demonstrate that the proposed approach can automatically and correctly identify cloud system configurations that match tenant-specific requirements or identify the group of tenants that have subscribed to a particular set of cloud features. In addition, the execution time of our proposed approach is only a small fraction compared to the existing approach using the formal method, e.g., FaMa, and the configuration space is also much smaller.

1. Introduction

Cloud computing consists of on-demand computing services with shared-access of configurable resources. There are three main cloud services: infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS). In addition, there are three common cloud deployment models: public, private, and hybrid. Cloud computing provides elastic services with lower cost in a pay-per-use model. Cloud services share similarities with Web services, but there are also distinct challenges that need to be addressed for cloud services exclusively [1]. Cloud services can be used by individual users or multiple tenants, where a tenant could

* Corresponding author.
E-mail addresses: ycao5@scs.carleton.ca (Y. Cao), chlung@sce.carleton.ca (C.-H. Lung), ajila@sce.carleton.ca (S.A. Ajila), xiaolin@cmail.carleton.ca (X. Li).

https://doi.org/10.1016/j.future.2018.12.025
0167-739X/

be an organization with a group of users sharing the same software system.

Multi-tenancy is an emerging technique that enables the cloud providers to achieve a large scale of resource sharing. Multi-tenancy can be realized for IaaS, PaaS, or SaaS. This paper presents a generic framework that can be used to support cloud service operations and management. This research, however, does not address actual deployment. Our approach has been applied to case studies in SaaS. Users subscribed to SaaS can access applications over the Internet on subscription basis. Concrete examples include email, customer relationship platform (CRM) to support management of sales, marketing, etc., in a central location [2], and a weather system [3].

The problems associated with developing, deploying, and maintaining cloud computing, particularly multi-tenancy, are primarily caused by resource sharing, a large number of possible configurations [4–6], and various constraints such as cloud type or location. For SaaS, typically, a pre-packaged and configurable system [7] is provided to tenants, rather than the traditional on the premise model. In addition, there are difficulties of managing the tradeoffs over various non-functional requirements, such as performance, security, privacy, availability, and cost. As a result, the development of a multi-tenant application is much more difficult than the development of a single tenant application [6,7]. It is impractical to rely on tenants to make decisions on partitioning their applications and selecting appropriate configurations that satisfy tenant-specific requirements, constraints, and preferences. And, doing it manually is time consuming and error prone even for the professionals (e.g. cloud service providers), especially when the configuration space and the number of tenants is large. Formal methods, e.g., [8–10], have been proposed to solve the problem. However, formal methods typically are limited to small scale and the processing time usually is high.

The objective of this paper is to address the problem of large variations and complex configurations [5,11,12]. Particularly, the goal is to develop mechanisms to support automatic multi-tenant software features analysis and matching for efficient deployment and operations in the cloud. The focus of this paper is on the matching between the tenant-specific requirements and the SaaS features managed by the cloud provider. The importance of automatic matching between tenant-specific requirements and SaaS features cannot be over emphasized. To the best of our knowledge, there is little research reported in the literature to facilitate automatic matching between desirable tenant-specific configurations and SaaS features.

In our previous work [13], we presented the preliminary idea to support matching of SaaS features with features requested by multiple tenants using Yfilter [14]. Yfilter has been systematically evaluated in various applications in the open source community. Further, Yfilter is scalable to a very large number, e.g., 50,000 queries which are equivalent to features in our problem. In this paper we extend the scope to identify the group of tenants that have subscribed to a particular set of cloud features and more experiments for further validation and analysis using a larger set of features. In addition, a thorough analysis of the technical concepts is presented.

The main contributions of this paper are summarized as follows:

- The introduction of a reference framework used for constraint-based multi-tenant feature management with focus on SaaS.
- The novel application of XML-based techniques for cloud services, including (i) encoding the feature models for the SaaS system and tenant-specific requirements or preferences with XPath representations [15], and (ii) the application of Yfilter [14] technique to achieve three things:
  - Automatic selection of SaaS features from a possible large set of options and complex constraints that meet a tenant's requirements. This step can help feature based configuration for each tenant and can be used for potential further deployment.
  - Automatic identification of tenants that have subscribed to a set of SaaS features. This step can be used if some SaaS features or associated business policies are updated.
  - Reduction in the amount of configuration space and the computation time.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 presents an overview of the proposed reference framework. Section 4 describes the automatic matching for multi-tenant feature configurations using the XML-based filtering techniques. Section 5 illustrates case studies based on the proposed approach. Section 6 describes the tool used in the experiments and case studies. Finally, Section 7 concludes the paper and discusses some future research directions.

2. Related work

This section describes three related research areas: feature modeling, existing research on multi-tenant software systems, and techniques in XML that are related to our research. A discussion and summary of those techniques that are related to our research is presented at the end.

2.1. Feature modeling and related work

Feature model [16] is a representation of visible and useful aspects of a software product or software product lines. It is a visual representation by means of feature diagrams and it is widely used in the design and development of Software Product Lines (SPLs) [17,18]. Feature is a noticeable user visible aspect, quality attribute or functional characteristic of a software product [19]. Basic feature models have a reduced set of syntax that describes the relationship between a "parent" feature and a "child" feature using a set of operators [17]. Basic feature model operators are *mandatory* – the child feature is required; *optional* – the child feature is optional; *OR* – at least one [sub] feature must be selected; and *XOR* – one of the [sub] features is selected. A feature diagram is tree-like structure with special operators, e.g., *AND*, *OR*, *XOR*, and constraints that define features and their valid combinations in configurations [20]. In this paper, we have used feature modeling to capture both cloud provider features (i.e., SaaS) and tenant-specific requirements.

Feature modeling has been applied to SaaS to manage configuration and customization of service variants [3,21,22]. In particular, the research work by [23] extended variability modeling to customize the SaaS application and guide the SaaS provider for service deployment. Feature modeling was also used by [23] and others [24–26] to select the best cloud service (PaaS or IaaS) provider for a given application. The tool FAMILIAR proposed by [23] is used to merge feature models of all available PaaS solutions into an aggregated feature model. However, the authors did not present how to automatically identify a cloud provider based on feature models. Wittern et al. [24,25] utilized service feature modeling (SFM) to represent service design concerns and their potential combinations. They also adopted a method to rank service design alternatives based on stakeholder preferences. Their ranking approach is built upon multi-criteria decision making (MCDM) which was also adopted for selecting cloud services by [27]. But, the processing time for configuration ranking could be high and the interpretation of the ranking could be challenging [25].

Authors in [28] investigated three approaches for the development and management of multi-tenant SaaS software: application-based, feature-based, and a hybrid approach. The paper presents a theoretical analysis by comparing the number of instance types that is generated by each method. Based on their comparison, the feature-based approach generates fewer runtime instance types.

The research work by [8] developed a tool called FaMa [29]. The tool is used to convert a feature model to a formal representation, e.g., Boolean satisfiability (SAT), constraint satisfaction problem (CSP), or binary decision diagram (BDD), and it uses a corresponding solver to perform analysis on the feature model. Although FaMa is a good tool for analyzing feature models, it generates a large number of configuration options that needs further refinement. The approach has not been demonstrated to be able to analyze medium to large size feature models in a reasonable time, which is a concern for its scalability. Further, a formal framework called FLAME based on Z specification language [30]. FLAME is used for specifying the semantics for feature models, i.e., FaMa, and that for various variability modeling languages. The framework supports filtering of an SPL which can be used to select a set of features or products of a SPL for a given configuration.

Hajlaoui et al. [9] also investigated three algorithms based on graph edit distance which is followed by ranking the QoS as a weighted graph matching problem. The experiments were only conducted for a small number of features. In addition, the results for precision and recall could spread to a wide range. Recently, Xiang et al. [10] introduced an approach that combines a many-objective optimization algorithm and two SAT solvers to generate the optimal feature selection problem. One solver is used to repair invalid configurations when dealing with many-objective optimal feature selection problem, the other one is introduced to produce dissimilar solutions. As a result, both the number of features and constraints are reduced significantly. Hence, the computation time is dramatically reduced.

2.2. Multi-tenancy and related work

A *tenant* is defined as a cloud customer consisting of a group of users (or software developers)) sharing the same view of a software system. Multi-tenancy is defined as multiple tenants or processes or organizations sharing resources (both physical and virtual) while they remain logically independent. Multi-tenant (or multi-tenancy) is a way of providing to each tenant a "share" of the instance of an application (or service) that is isolated from other "shares" of the same instance with respect to performance, security, and data privacy. Multi-tenancy is different from multi-user or multi-instance [4]. In multi-tenancy, compared to multi-instance, each tenant may have high degree of configurability of the software system and the number of tenants can be very large. Some key features of multi-tenancy include hardware resource sharing, large configuration space, and a limited number of instances of a single application or database system. In addition, there is the possibility of having different workflows for the same application for different tenants [4], and even different versions may be needed for the same tenant [6]. As a result, a number of challenges arise such as scalability, operational costs, flexibility, and security. These challenges are key considerations for multi-tenant SaaS systems development, deployment, and management [6].

A great deal of research has been conducted on this topic. However, most of the existing research on multi-tenant SaaS systems has focused on shared infrastructure (i.e., virtualization techniques), security management, data architecture, and middleware extensions [31]. On the other hand, the challenge due to a large amount of tenant configuration space for multi-tenancy has not been adequately explored.

The work by [4] described the fundamentals of the multi-tenant approach and compared it with the multi-user and multi-instance approaches. Further, the authors presented the main challenges for the multi-tenant approach, which includes the large configuration space. In addition, the authors also identified four configuration types: layout styles, general configuration, file I/O, and workflow. Different types of configuration enable the tenant to have a customized user-experience as if the system is a dedicated environment for the tenant. Both [4] and [31] adopted the 3-tier architecture (authentication, configuration, and database) to support multi-tenancy.

Walraven et al. [6] discussed the multi-tenant systems and the challenges associated with the systems. The authors advocated the essentiality of automatic transformation or mapping of tenant-specific feature configurations and SaaS application features. The authors concluded that there was a lack of methodical support for the development of multi-tenant applications and hence proposed a service line engineering approach in support of co-existing tenant-specific configurations. This work, however, only focuses on the methodology and falls short of demonstrating any automatic transformation technique.

In [32], the authors investigated the scalability challenge in customizing tenant-specific features for SaaS. They considered two stakeholders – service customer (tenant) and SaaS service provider – and investigated the impact of feature interactions on scalability for a multi-tenant SaaS system. Feature mapping is one of the key tasks in their framework, which is based on configuration file used to describe the required features.

In [28], the authors investigated feature placement algorithms for multi-tenancy for cost-effective resource allocations. The algorithm is used to determine where to place feature instances and the amount of resources for them. Random feature models were generated to evaluate the optimization algorithm based on an integer linear problem solver and a heuristic algorithm. Based on the evaluation, using feature instances increased the achievable level of multi-tenancy.

Garćıa-Galán et al. [33] described the increasing complexity of highly-configurable services (HCS) for XaaS due to variabilities of the decision space. The critical tasks involved for HCS include description, discovery and selection of services. The authors introduced SYNOPSIS, a new domain specific language, that captures the decision space based on feature modeling. SYNOPSIS can be used for validity checking for HCS and selection of the most suitable configurations with respect to user requirements. Selecting the best configurations is similar to the theme of our paper. However, we utilize XML and its related techniques to represent and select HCS. The next subsection briefly describes XML and its related techniques that are used in this paper.

A few other approaches related to multi-tenant SaaS have been published in the literature. Alwis and Gamage [34] proposed a model-driven approach and used a UML 2.0 based profile to develop SaaS applications. Dutta and Gupta [35] studied the customization issues in SaaS applications and used XML to store tenants' customization data in a database. The approach, however, does not address feature selection. Espadas et al. [36] developed a tenant-based resource allocation model for multi-tenant SaaS applications. Walraven et al. [37] designed a middleware layer that supports dependency injection and tenant data isolation using Google App Engine to handle customization and multi-tenancy. Tsai et al. [38] proposed sub-tenancy architecture (STA) that allows tenants to provide services for subtenant developers for SaaS development and customization. Feature requirements, modeling, and selection are central to the application development. The authors advocated crowdsourcing for feature implementation and selection.

2.3. XML, JSON, XPath, and XML filtering techniques

XML [39] has been widely used to represent structured information in a machine-readable format. It benefits from interoperability and eliminates differences in proprietary protocols between networks, operating systems, and computing platforms. JSON [40] shares many similarities with XML and has become a popular data exchange format. However, the use of JSON is primarily limited to data exchange between servers and clients, especially on the browser side due to its lightweight and frequent execution. XML, on the other hand, is a language and has a rich set of features associated with it [41]. Further, XML, if running on the server side, does not reveal much difference in performance compared to JSON. In this work we used XML to encode tenant-specific requirements.

More importantly, the choice of XML over JSON is based on the fact that XML-based techniques are more closely related to our research than JSON and that XML-based methods have been used for feature modeling. XPath, a query language is part of XSLT standard and it is used to navigate using path expression through the elements and attributes of an XML document. XPath is also a language for retrieving, selecting, and filtering information from an XML document. XML filtering is a well-established technique in XML databases and XML publish/subscribe systems. XML filtering can identify matching between an XML document d and an XPath feature (query) representation F. The matching operation is complex as it needs to identify whether d has an isomorphic tree-like structure and values defined in F. Further, there are nested query relationship and complex operators, e.g., "*" (wildcard), '//' (ancestor/descendant operator), and "[]" (predicate).

The main benefit of using XPath feature modeling is its ability to support expressions of tree-like structure of feature diagrams. This characteristic can be used to avoid enumerating of all possible combinations of features, which is time-consuming and error-prone. There are a number of XML filtering techniques reported in the literature. They include Yfilter [14], Afilter [42], Bfilter [43], Gfilter [44], Hfilter [45], Pfilter [46], Sfilter [47,48], and Wfilter [49]. The key difference for those filters is the data structure used. For instance, Yfilter makes use of a nondeterministic finite automaton (NFA) in a top-down fashion, Afilter uses a directed graph, Gfilter adopts a tree-of-path coding scheme for query processing via a bottom-up approach, and Bfilter builds on top of Yfilter but delays the matching process until a branch point is matched. A comparison of those filters can be found in Dai, et al. [43]. Among them, Yfilter is the representative technique, because it has been thoroughly investigated and broadly evaluated against other filtering techniques by multiple research groups in the open source community. Moreover, Yfilter is open source and easier to understand, as it makes use of the tree structure, and it has been demonstrated that it can handle a large number of queries (e.g., 50,000) [14].

Additionally, our choice of XPath is based on the fact that it is a key element in Yfilter suite and is robust and scalable. An example of XML-based feature modeling tool is XFeature [50], used for representing feature diagrams, modeling of product families, and instantiating applications from the model. However, XFeature does not support the automated analysis of feature diagrams, which is crucial for a large number of configurations. Another example similar to our method is the approach presented in [12] which adopts XPath to support view specification based on feature diagram. The authors also integrated XPath with the SPLOT [51], an open source system for creating, editing, and sharing feature diagrams.

2.4. Discussion

Multi-tenancy has the potential to reduce cost, as multiple users share the same application instance and resources. However, there is a lack of support to effectively manage different tenant requirements or preferences [52]. Consequently, two approaches are common in practice: one-size-fits-all approach [6] or an application-based approach [28]. The one-size-fits-all approach may not work well for tenant-specific requirements, whereas the application-based approach may generate a large number of application instances. On the one hand, multi-tenancy comes with many variations such as SLAs (Service Level Agreements) [53], different constraints, or even the workflow may be different for different tenants. In addition, versioning may be needed for the same tenant on top of those variations. As a result, the configuration space can grow rapidly. On the other hand, techniques based on formal optimization-based methods, e.g., SAT, CSP, or BDD [17,54], have yet to demonstrate the efficiency for a large number of configurations that the multi-tenancy systems often may exhibit. In addition to the optimization-based service selection methods, two other main approaches are MCDM-based and logic-based [1]. The configuration space issue still exists for those approaches.

Therefore, as the feature configuration space grows rapidly and the number of tenants and requirements increase, there is a need for an efficient and scalable approach to automatically transform and map SaaS provider features to tenant-specific requirements. Furthermore, feature modeling has been adopted by various researchers for multi-tenant SaaS system. Feature modeling has been extensively used for variability management, which has a direct impact on development and deployment complexity, performance, and consistency [28]. Also, XML techniques have been used by various researchers where scalability is critical, e.g., publish/subscribe systems [14,42–44] and databases and information sciences [55–61].

In this research work, we adopt feature modeling to capture commonalities and variabilities of SaaS systems and tenant-specific requirements, and use XML to encode tenant-specific features. In comparison to other approaches especially the formal methods, e.g., [17], the configuration space of our Yfilter-based approach is much smaller, because common paths for features in a feature diagram are shared. In other words, Yfilter aggregates common paths rather than enumerates all possible feature options. As a result, the processing time is also low, which is suitable for large numbers of features and tenants.

3. Constraint-based cloud deployment framework

This section presents an overview of our constraint-based multi-tenant approach for supporting cloud deployment. The framework is a modification of the method presented in [6]. Contrary to their approach, our framework is generic and applicable to the three types of cloud services (i.e., IaaS, PaaS, and SaaS). In addition, our framework is not geared towards application services as is the case of their approach; instead, our proposed approach is designed to be used to *support* the deployment and management of software systems in the cloud. In other words, our framework is not just to support SaaS deployment, but a set of support mechanisms for cloud deployment and management for cloud services (IaaS, PaaS, and SaaS) which can be represented with a high-level abstraction of features. The current focus of the paper is on SaaS and the framework is applied as a case study to support SaaS deployment using XPath feature representation and Yfilter technique for XML matching (cf. Sections 4 and 5).

As depicted in Fig. 1, the framework consists of four processes (shown in round rectangles): Cloud Services Analysis and Configuration Management, Tenant Requirements Analysis and Configuration Management, Cloud Service Deployment, and Shared Services and Configuration Management. Fig. 1 also shows the main activities, artifacts, input/output, and data dependency.

The Cloud Services Analysis and Configuration Management (top left in Fig. 1) models cloud services/products represented

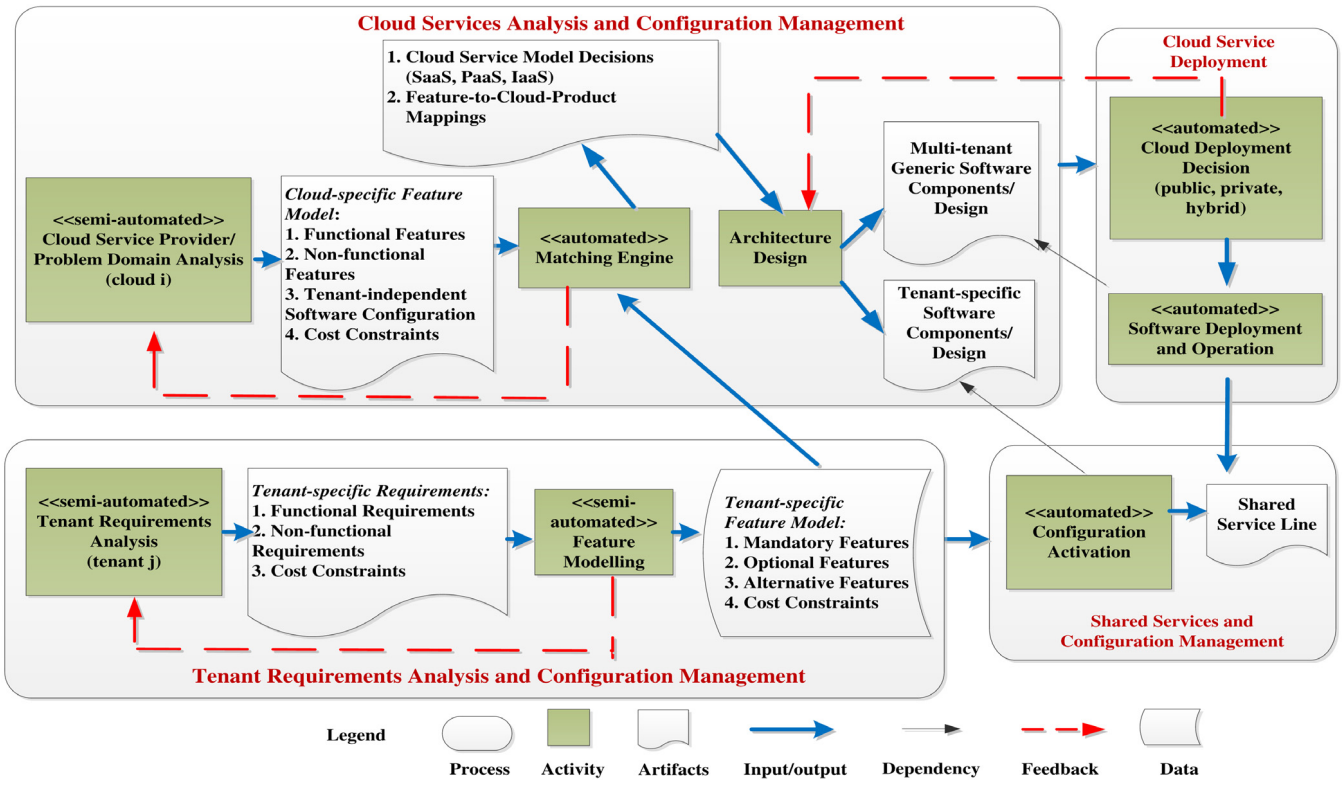

Fig. 1. Constraint-based cloud deployment framework.

using feature diagrams. The starting point is the cloud service analysis and problem domain analysis. A cloud specific model is generated and the outputs include functional and non-functional features, tenant-independent software configurations, and cost constraints.

Similarly, the Tenant Requirements Analysis and Configuration Management process starts with Tenant Requirements Analysis to identify features and constraints. A tenant-specific feature model is then generated as a result of Feature Modeling.

Following that, the Matching Engine performs the matching activity, which is a key focus proposed in this paper, between the cloud and tenant-specific features. The primary outcomes of the matching process are cloud services model decisions and the feature-to-cloud-product mappings.

The "cloud services model decisions" (top in Fig. 1) are presented to the user for decision making in terms of the choice of cloud provider, type of cloud service, constraints, costs, etc. In addition, the user has the freedom to modify the initial requirements or add more requirements. This is the major reason why both the domain and tenant requirements analysis are semi-automated. The idea is to give freedom of choice in cloud provider and cloud service to the user based on cost constraints.

Architecture Design deals with components that are generic or specific. Generic components are shared among a large number of tenants, while specific components are tailored to one or a small number of tenants, based on the feature analysis.

Note that this paper focuses on two processes: Cloud Services Analysis and Configuration Management and Tenant Requirements Analysis and Configuration Management in support of feature-based configuration and cloud deployment. Between these two processes, the Matching for configuration is the key concern of this paper. The actual deployment related activities, i.e., Cloud Deployment Decision, Software Deployment and Operation, and Configuration Activation, of the cloud service is beyond the scope of the paper.

4. Automatic XML-based matching

This section presents a summary of an XML-based matching technique in support of SaaS configuration. The approach consists of five steps:

1. Perform feature modeling for the SaaS system
2. Encode SaaS features with XPath representations
3. Perform tenant-specific feature modeling
4. Encode the tenant-specific features with XML
5. Perform XML filtering and matching operation with Yfilter. The filtering and matching operation can be used for two different types of applications:

    a. Type I: Identify the matched SaaS system features based on the tenant-specific requirements and constraints.
    b. Type II: Identify all interested tenants for a particular set of the SaaS services, type, or applications.

In what follows, we first briefly explain steps 1 and 3. Next, we describe steps 2 and 4 that deal with encoding feature diagrams using XML techniques, which is followed by an introduction to the Yfilter filtering and matching technique.

4.1. Perform feature modeling

Both steps 1 and 3 are the same as the traditional feature modeling or SPL process. The feature diagram for a tenant (the result of step 3, which includes tenant-specific constraints) typically is a subset of that of the SaaS systems. Tenant-specific feature model, hence, can be easily obtained by modifying the feature model of the SaaS application.

In SPL, various relation types can be modeled: mandatory, optional, XOR (only one feature can be selected), OR (at least one

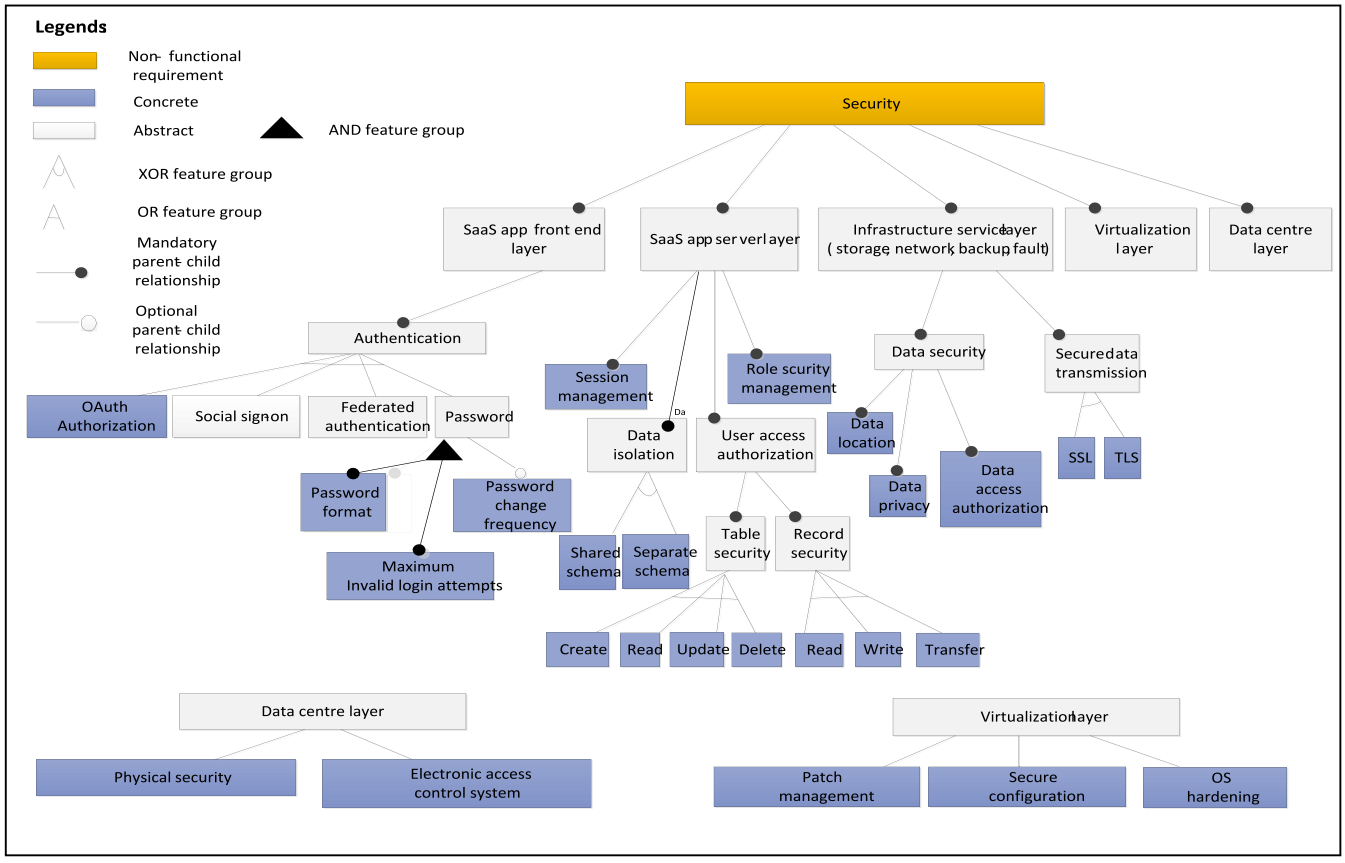

Fig. 2. Feature diagram for the security aspect of Saleforce.com CRM.

feature is selected), AND (each feature must be selected), *includes*, and *excludes* [21].

Fig. 2 presents a simplified feature diagram developed for the security aspect for the Saleforce.com customer relationship management (CRM) system as an example. There are five main features in Salesforce.com CRM: SaaS app front end, SaaS app server, Infrastructure service, Virtualization, and Data center. A mandatory relationship is illustrated between the SaaS app server layer and the Session management, Data isolation, User access authorization, and Role security management. Password change frequency shows an example of an optional relationship for Password. Data isolation can be either Shared or Separate schema for an XOR relationship. The OR relationship is illustrated in the Authorization feature, e.g., at least one authentication method, Social sign on, Federated authentication, or Password, must be selected.

4.2. Apply XML-based matching technique for features

Steps 2 and 4 deal with encoding feature diagrams using XML techniques, which will be used for the Yfilter filtering and matching technique.

As mentioned earlier in Section 4, there are two potential types of applications for our proposed approach. This section uses a Type I application for illustration. An example for Type II application is presented in Section 5 Case Studies.

To meet the objective of Type I application, the authors in [17] proposed to use the FaMa technique [29]. In their approach, the feature diagram is translated into logical representations and mapped to a corresponding solver. Then, the result is mapped to the feature domain again to present it to users. The choice of the solver depends on the type of analysis required for the feature diagram, e.g., a SAT solver for Boolean satisfiability checking. The FaMa approach has yet to demonstrate how to overcome some limitations, such as: (1) the space complexity for the medium and large size feature diagrams can grow exponentially; and (2) the logical representation for feature diagrams is not straightforward because of the conversion between the feature diagram and the logical representation.

In this paper, we use XPath and XML representations to encode the feature diagrams for a SaaS system and tenants, respectively. Following that, we apply Yfilter to identify tenant-specific feature-based configuration.

4.2.1. Encoding feature diagrams with XML and XPath

Nodes in a feature diagram correspond to location steps of an XPath feature representation. Each location step contains an axis, a node test, and zero or more predicates. An axis specifies the hierarchical relationship between nodes. For example, the parent/child ("/") operator or ancestor/ descendant operator ("//").

A path starting from the root node and ending at a leaf node can be encoded as a single XPath feature representation. For example, the path starting from Security and ending at DataLocation, as depicted in Fig. 2, is encoded as an XPath feature representation:

/Security/InfrastructureServiceLayer/DataSecurity/DataLocation.

The flexible grammar of XML and XPath is used to represent various relationships captured in feature diagrams. Table 1 lists the XML and XPath query representations for feature relationships, which is briefly explained as follows.

For a *mandatory* feature in a feature diagram, the corresponding XML and XPath representations are straightforward, as depicted in Table 1 . For an *optional* feature, the keyword *select* is used, which is coupled with a Boolean value 1 (or 0) representing the presence (or absence) of the feature.

Next, an *OR* feature group specifies a group of features from which at least one feature must be selected. Hence, the OR feature

Table 1
XML and XPath representation feature.

Feature relationship	XML	XPath feature representation
Mandatory	<featureName/> Example: <Security><SaaS app front end layer/></Security>.	/featureName Example: /Security/SaaS app front end layer/.
Optional	<featureName select="0"/>, <featureNameselect="1"/>. Example: <Password change frequency select="0">, <Password change frequency select="1">.	/featureName[@select="0"], /featureName[@select="1"]. Example: /Password change frequency[@select="0"], /Password change frequency[@select="1"].
OR	<featureName><branch$_1$/><branch$_i$/> <branch$_N$/>< /featureName> Example: <Table security><Create/><Read/></Table security>.	/featureName/branch$_1$, /featureName/branch$_i$, /featureName/branch$_N$. Example: /Table security/Create, /Table security/Read, /Table security/Update, /Table security/Delete.
AND	<featureName><branch$_1$/><branch$_2$/> <branch$_N$/></featureName> Example: <Password><PasswordFormat></PasswordFormat> <MaxInvalidLoginAttemps num="3"/></Password>.	/featureName[branch$_1$][branch$_2$]/branch$_N$ Example: /Password[PasswordFormat]/MaxInvalidLoginAttempts[@num≤3].
XOR	<featureName><branch$_i$/></featureName> Example: <DataIsolation><SharedSchema/></DataIso lation>.	/featureName/branch$_1$, /featureName/branch$_i$, /featureName/branch$_N$. (Notes: Post-processing is required for XOR.) Example: /DataIsolation/SharedSchema, /DataIsolation/SeparateSchema.
Attribute constraint	<featuerName attribute="value"/>, <featureName>someValue</featureName>. Example: <Data security><Data location>Canada</Data location></Data security>.	/featureName[@attribute="value"], /featureName[text()="someValue"]. Example: /Data security/Data location[text()="Canada"].

group is represented as multiple XPath feature representations that are selected. An *AND* feature group consists of features from which every feature must be selected. Therefore, a tree-structure XPath feature representation (using the "[]" operator) can be used to encode the semantics of the AND relationship. The example for AND in Table 1 shows that branch$_1$, branch$_2$, and branch$_N$ are all selected.

The *XOR* feature group consists of features from which only one can be selected. A two-step process is required to handle the XOR relationship: (1) The features that are covered with XOR are represented as a set of XPath feature representations; and (2) a post-processing phase is applied to verify the exclusive relationship among features. Lastly, a feature diagram may include *attribute* constraints which are represented as value-based predicates in XPath feature representations. In Table 1, the XML and XPath examples represent the corresponding part of the feature diagram as depicted in Fig. 2. For illustration purposes and due to the space limitation, only a fragment of XML and XPath feature representations are shown in the examples. To handle the *includes* and *excludes* relationships, which idea is similar to *optional* feature, select $= 1$ (used for *includes*) and select $= 0$ (used for *excludes*) together with the actual name. The difference is that additional post-processing is required after matching to verify the correctness of the *includes or excludes* relationship, as the attributes could be for different nodes for the features.

4.2.2. Applying Yfilter matching technique

Our proposed matching approach using Yfilter, step 5, can be applied in two different ways. This section first describes the application of identifying the SaaS cloud features that match tenant-specific requirements, constraints, and/or preferences. For this application, after encoding the cloud-side feature diagram as a set of XPath feature representations, these XPath feature representations are organized as a nondeterministic finite automaton (NFA), e.g., Fig. 3, a formal model adopted by Yfilter. An XML document which contains the configuration for each tenant is evaluated against the NFA using Yfilter. The generated results from Yfilter are a set of SaaS cloud features that satisfy the tenant's requirements and constraints.

Note that for Type I application, the matching operation is not required every time a tenant specifies requirements in the analysis stage if the feature model of the SaaS system is visible to tenants. In this case, the SaaS features can be presented to tenants to choose from. Rather, the step can be used when there are complex tenant-specific constraints, e.g., features based on various feature operators: AND, OR, XOR, *includes*, *excludes*. This step can also be useful to identify feature-based configuration for tenants or to support potential downstream deployment based on identified features.

On the other hand, Type II application can be used to identify all tenants that have subscribed to a certain set of SaaS features. In this case, the encoded XPath for SaaS will be evaluated against the aggregated XPath features for multiple or all tenants. The application becomes very similar to the publish/subscribe system where the filtering operation has been used to identify all subscribers (i.e., tenants in our case) that are interested in a particular publication document (e.g., a set of SaaS system features that may need to be updated).

Fig. 3 depicts an example of three features expressed in XPath and the corresponding NFA used in Yfilter. The NFA tree consists of three features F0, F1, and F2. Each circle in the figure represents a state, and double circles denote an accepting state; such states are associated with the features they represent, e.g., text() = "Canada". A directed edge with a solid arrow represents a state transition. The symbol on an edge represents the input that triggers the transition. The special symbol "$*$" is a wildcard which matches any element. The symbol "ε" is used to mark a transition that requires no input. The operator "//" in an XPath feature representation is mapped to a combination of a "ε" transition and a self-loop transition with a "$*$" input tag. The operator "//" results in a state being visited many times.

Yfilter processes an XML document by searching from the root of the NFA and tracks all matched transition states for each start tag. There is a selection operator for each accepting state to handle the value-based predicate in the corresponding path. For a tree-structure XPath feature representation, Yfilter splits such a tree

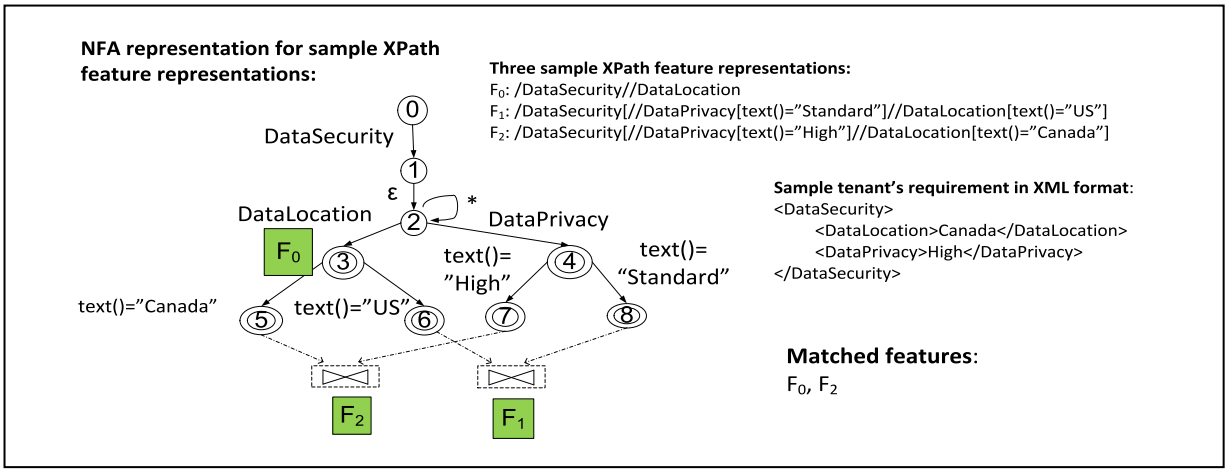

Fig. 3. Three sample XPath feature representations and a sample Tenent's requirements in XML format.

feature representation into a set of simple path expressions and applies a join operation after selections. The join operator finally validates the correctness of the tree structure.

The aggregated XPath feature representations in Fig. 3 can be viewed (top right) as the feature set for the cloud system and the sample XML fragment (middle right) represents a set of tenant-specific requirements. In this example, the matched features are F0 and F2 (bottom right) which satisfy the tenant's requirements.

A key benefit of using an NFA-based approach is the tremendous reduction in the space size. The space complexity of our approach is O(n), where n is the number of nodes in a cloud-side feature diagram. For time complexity, although the NFA can result in fast growth due to an increase in the number of states when we match a recursive tag element in an XML document with a descendant operator ("//") in an XPath, the average performance of Yfilter is still sufficiently fast as reported in [14]. For example, the processing time of Yfilter is only around 30 ms when the number of queries is slightly larger than 50,000.

4.3. Validation of the proposed approach

It has been proven that Yfilter can correctly generate the matching results based on formal NFA for XPath representations and correctly identifies matched subscribers encoded in XML for the publish/subscribe application domain [14]. It has also been adopted for various applications in information processing. In order to apply Yfilter and its related technologies to our problem domain, we need to map two critical components: aggregated XPath representations (conceptually represented in NFA) and incoming XML messages. The analogical mapping from the SaaS problem to publish/subscribe systems can be realized as follows:

- The aggregated set of XPath features for an SaaS system are mapped to the aggregated subscriptions in publish/subscribe systems
- Features encoded in XML for each tenant are mapped to publisher's document in publish/subscribe systems

In other words, these two problem domains are analogous: when a publication document is available in a publish/subscribe system, Yfilter is used to identify matched subscriptions (or subscribers) against the aggregated subscription tree. Similarly, in the Type I application, given a tenant's feature set, Yfilter is used to identify a set of matched features against an aggregated feature tree provided by the SaaS provider. The matching operation for these two application domains is conceptually identical; hence Yfilter also can generate correct matching results for our proposed application.

The number of features (or queries in the publish/subscribe domain) used in some XML filtering technique is much larger than the numbers of features reported in the traditional SPLs. In comparison, the authors [54] showed an example of diagnosis of over 5000 features which is considered a very large number for the traditional SPLs for $\approx$ 50 s. (Note that diagnosis of features may require different computational cost, but the result reveals the range of time it needs.) For multi-tenant SaaS systems, the number of tenants alone can be in thousands or higher, and the total number of features for real systems could also be in thousands, e.g., 10,000 in [5]. Configuration complexity management and scalability are two key challenges for large systems [11,12,62]. For instance, the security feature, as depicted in Fig. 2, can be integrated into the feature diagram, as shown in Fig. 4, which can generate a large number of combinations even for a simple case study, especially when using a formal solver based on SAT, CSP, or BDD, such as FaMa [29].

Since tenant requirements and preferences usually change over time [52], and new tenants can join or tenant groups can evolve, the mapping algorithm may need to be executed frequently. Therefore, an efficient and scalable automatic mapping technique is essential in SaaS deployment [6]. The proposed application of Yfilter, related techniques, and the use of NFA can meet these challenges.

5. Case studies

We have conducted experiments using our proposed approach on a subset of features for Amazon EC2 service [63] and Salesforce.com CRM system [2]. The first case study illustrates the Type I application. The second case study is a variation of the first one and it shows that specific constraints can be considered with the proposed XML matching technique. Examples of constraints include identification of a cloud provider that supports certain features, public versus private clouds, or specific location information for the security concern. Finally, the third case study is an example of Type II application.

5.1. Amazxon web services—identifying tenant-specific SaaS features

In [17], the authors modeled Amazon EC2, EBS, S3 and RDS with feature diagrams as shown in Fig. 4. The feature diagram was analyzed with FaMa [29].

Although only functional features are considered, this small feature diagram yields 1758 different valid product options generated

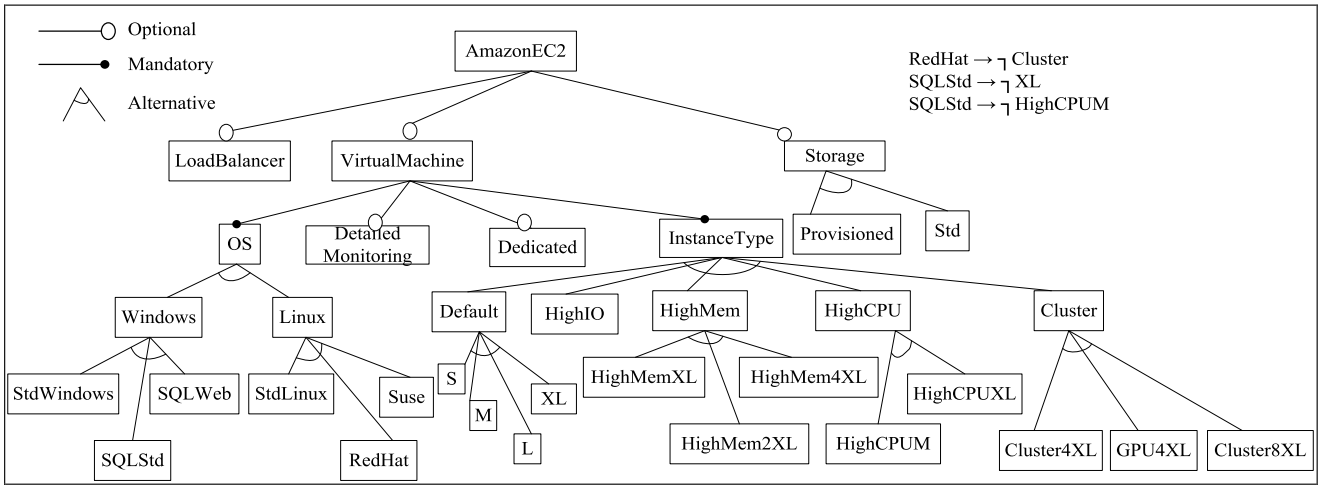

Fig. 4. Feature diagram for Amazon EC2 [17].

Table 2
XPath feature representation for Fig. 4.

Feature id	XPath feature representations (Note that each path starts with /AmazonEC2)
1	/LoadBalancer[@select="1"]
2	/LoadBalancer[@select="0"]
3	/VirtualMachine[@select="0"]
4	/VirtualMachine[@select="1"]/OS/Windows[text()="StdWindows"]
5	/VirtualMachine[@select="1"]/OS/Windows[text()="SQLStd"]
6	/VirtualMachine[@select="1"]/OS/Windows[text()="SQLWeb"]
7	/VirtualMachine[@select="1"]/OS/Linux[text()="StdLinux"]
8	/VirtualMachine[@select="1"]/OS/Linux[text()="RedHat"]
9	/VirtualMachine[@select="1"]/OS/Linux[text()="Suse"]
10	/VirtualMachine[@select="1"]/InstanceType/Default/S
11	/VirtualMachine[@select="1"]/InstanceType/Default/M
12	/VirtualMachine[@select="1"]/InstanceType/Default/L
13	/VirtualMachine[@select="1"]/InstanceType/Default/XL
14	/VirtualMachine[@select="1"]/InsanceType/HighMem[@RAMRightBound=4]
15	/VirtualMachine[@select="1"]/InstanceType/HighMem[@RAMRightBound=8]
16	/VirtualMachine[@select="1"]/InstanceType/HighMem[@RAMRightBound=16]
17	/VirtualMachine[@select="1"]/InstanceType/HighIO
18	/VirtualMachine[@select="1"]/InstanceType/HighCPU[@cuRightBound=3]
19	/VirtualMachine[@select="1"]/InstanceType/HighCPU[@cuRightBound=16]
20	/VirtualMachine[@select="1"]/InstanceType/Cluster/Cluster4XL
21	/VirtualMachine[@select="1"]/InstanceType/Cluster/GPU4XL
22	/VirtualMachine[@select="1"]/InstanceType/Cluster/Cluster8XL
23	/VirtualMachine[@select="1"]/DetailedMonitoring[text()="0"]
24	/VirtualMachine[@select="1"]/DetailedMonitoring[text()="1"]
25	/VirtualMachine/Dedicated[@select="0"]
26	/VirtualMachine/Dedicated[@select="1"]
27	/Storage[@selected="0"]
28	/Storage[@selected="1"][text()="Provisioned"]
29	/Storage[@selected="1"][text()="Std"]
30	/AmazonEC2/VirtualMachine[@select="1"]/InsanceType/HighMem/HighMemXL

by FaMa. Moreover, the execution time for identifying only one valid partial configuration from this feature diagram is 135 ms [17].

The execution time for the optimization operation is even more than one hour. As more features are considered, the matching task becomes far more complex. The high execution time using the approach in [17] is mainly caused by enumerating all possible configurations offered by a feature diagram one by one.

Using our proposed approach, the encoded XPath feature representation for Fig. 4 is listed in Table 2, which contains 30 features in total. There are three tenants used in the experiment. The requirements of tenant A are captured in an XML document, as shown in Fig. 5. Table 3 summaries the features specified for tenants A, B, and C.

The next step is to apply Yfilter to find the matched features provided by the SaaS for a tenant. As expected, the resulting features identified by Yfilter for tenant A, as depicted at the bottom of Fig. 5, are consistent with the features listed in Table 3. Results for tenants B and C are also correct, but are not presented here for brevity because they are similar to Fig. 5. In addition, Yfilter only takes around 5 ms for the matching operation. The measured processing time is given in Table 4. The processing time is much lower than the time taken using the method presented in [17] which is 135 ms for only one valid configuration without the optimization operation or ∼5600 ms with the optimization operation. The execution time grows exponentially to ∼32 000, ∼370 000 ms, and ∼1 h 30+ for two, three, and four configurations, respectively, with the optimization operation.

5.2. Saleforce.com — matching specific tenant constraints

This section presents another case study that is aimed to match specific tenant constraints. For instance, some features may be offered by different 3rd party providers. Our proposed approach

```
*** Required features for tenantA  in XML format: ***
<AmazonEC2>
    <LoadBalancer select="0"></LoadBalancer>
        <VirtualMachine select="1">
            <InstanceType>
                <HighMem RAMRightBound="8"></HighMem>
                <HighIO/>
                 <HighCPU cuRightBound="3"></HighCPU>
                <Cluster><Cluster4XL></Cluster4XL></Cluster>
            </InstanceType>
            <OS><Linux>StdLinux</Linux></OS>
           <DetailedMonitoring></DetailedMonitoring>
       </VirtualMachine>
       <Storage selected="1">Std</Storage>
</AmazonEC2>

*** The matched result is as follows: ***
 17  /AmazonEC2/VirtualMachine[@select="1"]/InstanceType/HighIO
  2  /AmazonEC2/LoadBalancer[@select="0"]
 18  /AmazonEC2/VirtualMachine[@select="1"]/InstanceType/HighCPU[@cuRightBound=3]
 20  /AmazonEC2/VirtualMachine[@select="1"]/InstanceType/Cluster/Cluster4XL
  7  /AmazonEC2/VirtualMachine[@select="1"]/OS/Linux[text()="StdLinux"]
 29  /AmazonEC2/Storage[@selected="1"][text()="Std"]
 15  /AmazonEC2/VirtualMachine[@select="1"]/InstanceType/
HighMem[@RAMRightBound=8]
```

Fig. 5. Requirements and matching results for tenant A.

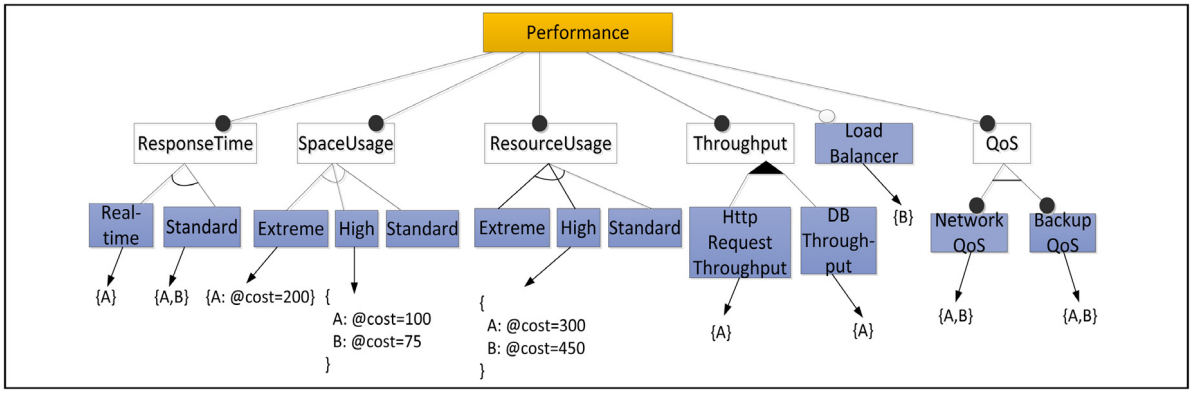

Fig. 6. Feature diagram for the performance aspect of Salesforce.com CRM.

Table 3
List of selected features for each tenant.

Tenant	List of selected features
A	Load Balancer: optional Virtual Machine: mandatory; Memory: RAM maximum 8; CPU: maximum 3; Cluster: 4XL; OS: Std Linux; Storage: Std
B	Load Balancer: opt ional Virtual Machine: mandatory; Memory: RAM maximum 16; CPU: maximum 3; Cluster: 4XL; OS: Std Linux; Storage: Provisioned
C	Load Balancer: mandatory Virtual Machine: mandatory; InstanceType: Default: Large; OS: Std Windows Storage: optional

can be used to identify features that satisfy a certain constraint. We use a subset of the performance feature of the Salesforce.com CRM system as an illustration. The feature diagram for the system is presented in Fig. 6.

Table 4
Processing time for the EC2 example.

Tenant	A	B	C
Matching time (ms)	5.4	5.8	5.7

This example also shows two features, A and B, which may represent (i) 3rd party service providers, e.g., PaaS, (ii) cloud-specific constraints, e.g., location for the security reason, or (iii) cloud type, e.g., public or private. For this simple example, we assume that A and B represent cloud providers. Some features are common in both providers, but some are unique to only one provider, e.g., the optional feature LoadBalancer is only available from provider B.

The first step is to encode SaaS features with XPath. All 15 encoded XPath representations for this simplified SaaS feature set are listed in Table 5. There are additional characteristics in this example. First, if the same feature is supported by multiple providers, the feature will appear for each tenant but with provider specific information. Take features F2 and F3 in Table 5 as an example. They are identical and available from both providers A and B. In this case, the same feature is repeated for each tenant in XPath representation, but with different provider information. Another point worth mentioning in this example is the features

```
<Performance>
  <ResponseTime><Real-time/></ResponseTime>
  <SpaceUsage><High cost="100"></High></SpaceUsage>
  <ResourceUsage>
       <High cost="300"></High>
  </ResourceUsage>
  <LoadBalancer select="1"/>
  <Throughput><HttpRequestThroughput/><DBThroughput/>
   </Throughput>
</Performance>
```

Fig. 7. A Tenant's features in XML: an illustration.

related to predicates. For instance, Features F5 and F6 are identical, except the cost values are different, $100 and $75 for provider A and provider B, respectively. The actual values are captured in the predicate format which can also be specified in a range.

The second step for the approach is to capture tenants' features in XML. From Fig. 7, the set of features specified for a tenant consists of Real-time feature, High SpaceUsage with cost = $100, High ResourceUsage with cost = $300, LoadBalancer, HttpRequest-Throughput, and DBThroughput.

The next step is to apply Yfilter for the mapping between the sample tenant's features (as shown in Fig. 7) and the set of features for providers A and B in XPath format (as depicted in Table 5). The matched features identified by Yfilter for the tenant are features F_1, F_5, F_7, F_9, and F_{11} (see Table 5). The features that cloud provider A supports (F_1: Real-time performance, F_5: High SpaceUsage with a price of $100, F_7: High ResourceUsage with a price of $300, and F_9: HttpRequestThroughput and DBThroughput) satisfy the sample tenant's features. However, only cloud provider B supports the optional LoadBalancer feature (F_{10} and F_{11}) in the sample tenant's requirements as shown in Fig. 7. The measured processing time for this case study is 4.5 ms.

This case study demonstrates that the matched results can be used to select a provider (a constraint in this case). Subsequently, the matched results can be transformed to the corresponding feature configurations in support of subsequent operations, which is out of the scope of the paper. The concept can be easily extended to choose a particular cloud type, e.g., public, private, or hybrid, or other specific characteristics that satisfy a tenant's specific requirements or constraints.

5.3. Amazon web services—identifying specific tenants for a set of cloud features

This section presents an example of the Type II application using the proposed approach. Specifically, the objective of this case study is to show that the proposed approach can be used to identify all tenants that are interested in or have subscribed to a certain set of SaaS features. The concept is similar to the XML publish/subscribe systems. This application can be useful for a scenario where a set of SaaS features are to be updated and there is a need to identify those tenants that may be affected for maintenance or management purpose.

The features presented in Table 3 are used to illustrate the application. The XPath representations generated by the Yfilter tool suite for those features (see Table 3) are listed in Table 6. Tenant A has 7 features, Tenant B, 7 features, and Tenant C, 4 features; and the total number of features is 18. The top 4 features are common to tenants A and B. Hence, there are 14 unique features in this example. In NFA-based XML filtering techniques, such as Yfilter, common features can be aggregated at one node in the NFA representation, which can save memory space and processing time. In other words, there are 14 nodes in the NFA for this example.

This experiment assumes that the SaaS provider offers those 14 unique features and wants to identify the tenants subscribed to a particular set of features. For instance, if the provider wants to identify the tenants for these two features:

/EC2/VirtualMachine[@select = "1"]/InstanceType/Cluster/Cluster4XL

/EC2/VirtualMachine[@select = "1"]/OS/Linux[text() = "StdLinux"]

Yfilter will return tenants A and B (the third and fourth features that are common to A and B). Yfilter also will return tenant C if load balancer is the feature of interest (both tenants A and B have not subscribed to this feature). XML-based filtering techniques can also support more complex XML documents in comparison to the traditional databases. For example, the entire 14 features can be used to identify tenants that are interested in each feature. Table 7 shows all tenants that are associated with each feature.

6. Tool support

XPath is the required format to represent queries (or features in our problem) for Yfilter. We adopt feature diagrams in the requirements management stage for an SaaS system. Hence, there is a need to convert feature diagrams to XPath representations in order to use Yfilter. This section first describes the tool we have developed and used to validate our proposed approach to the matching problem for multi-tenant SaaS systems. Following that, we describe case studies to demonstrate how those tools are used. In addition, some potential directions for tool integration are highlighted to support automation.

The Yfilter tool suite can be used to generate XPath queries from DTD (Document Type Definition). Therefore, we translate the feature model to a DTD file. This step currently is performed manually for proof-of-concept and evaluation of the XML related techniques for the multi-tenant matching problem. We have developed a tool to translate the DTD representation to generate all possible XPath representations corresponding to the feature model. The generated XPath representations are subsequently used by Yfilter for the matching purpose to validate the proposed approach and test the performance of our approach. When we need a new feature model, e.g., for a new tenant or modifications of an existing one, we just need to modify an existing feature model and DTD representations which can be then used to generate all possible XPath representations for the new feature model automatically.

On the other hand, the XPath query generator provided by Yfilter cannot take the relationship between elements into account when generating queries that are needed for feature diagrams, e.g., AND, OR, etc. Hence, we have also modified the XPath generator to create XPath feature representations that are consistent with the feature diagram. For instance, if element A and element B have an OR relationship, they will not appear in one XPath representation at the same time. Another example is if attribute C of element D is marked as mandatory, attribute C will always appears with element D in XPath queries.

To illustrate the tool support, we have combined two feature diagrams—Fig. 2 for the security aspect of the Salesforce.com CRM system and Fig. 4 features for Amazon EC2. The combined feature diagram has 71 features.

Fig. 8 depicts a partial list of the DTD representations for the combined 71 features as shown in Figs. 2 and 4. The complete list of DTD representations is presented in the Appendix. A feature starts with the keyword ELEMENT. For instance, Fig. 8 shows four top level features on the first row, which consists of Security in Fig. 2, and LoadBalancer, VirtualMachine, and Storage for AmazonEC2 in Fig. 4. LoadBalancer does not have any child, hence it is trailed with EMPTY. ATTLIST stands for attribute list. For instance, the attribute list for LoadBalancer (the 3rd row) shows that either 0 or 1 is needed for the feature, as LoadBalancer is an optional

Table 5
XPath representation for the feature diagram in Fig. 6.

id	XPath feature representations	Cloud provider
1	/Performance/ResponseTime/Real-time	A
2	/Performance/ResponseTime/Standard	A
3	/Performance/ResponseTime/Standard	B
4	/Performance/SpaceUsage/Extreme[@cost=200]	A
5	/Performance/SpaceUsage/High[@cost=100]	A
6	/Performance/SpaceUsage/High[@cost=75]	B
7	/Performance/ResourceUsage/High[@cost=300]	A
8	/Performance/ResourceUsage/High[@cost=450]	B
9	/Performance/Throughput[HttpRequest-Throughput]/DBThroughput	A
10	/Performance/LoadBalancer[@select="0"]	B
11	/Performance/LoadBalancer[@select="1"]	B
12	/Performance/QoS/NetworkQoS	A
13	/Performance/QoS/NetworkQoS	B
14	/Performance/QoS/BackupQoS	A
15	/Performance/QoS/BackupQoS	B

Table 6
Xpath representations for features in Table 3.

Tenant	XPath feature representations
A	/EC2/VirtualMachine[@select="1"]/InstanceType/HighIO / EC2/LoadBalancer[@select="0"] /EC2/VirtualMachine[@select="1"]/InstanceType/Cluster/Cluster4XL /EC2/VirtualMachine[@select="1"]/OS/Linux[text()="StdLinux"] /EC2/VirtualMachine[@select="1"]/InstanceType/HighCPU[@cuRightBound=3] /EC2/Storage[@selected="1"][text()="Std"] /EC2/VirtualMachine[@select="1"]/InstanceType/HighMem[@RAMRightBound=8]
B	/EC2/VirtualMachine[@select="1"]/InstanceType/HighIO /EC2/LoadBalancer[@select="0"] /EC2/VirtualMachine[@select="1"]/InstanceType/Cluster/Cluster4XL /EC2/VirtualMachine[@select="1"]/OS/Linux[text()="StdLinux"] /EC2/VirtualMachine[@select="1"]/InstanceType/HighMem[@RAMRightBound=16] /EC2/VirtualMachine[@select="1"]/InstanceType/HighCPU[@cuRightBound=16] /EC2/Storage[@selected="1"][text()="Provisioned"]
C	/EC2/LoadBalancer[@select="1"] /EC2/VirtualMachine[@select="1"]/OS/Windows[text()="StdWindows"] /EC2/Storage[@selected="0"] /EC2/VirtualMachine[@select="1"]/InstanceType/Default/L

Table 7
Identification of tenants for cloud features: An illustration.

XPath Feature	Tenant
/EC2/VirtualMachine[@select="1"]/InstanceType/HighIO	A, B
/ EC2/LoadBalancer[@select="0"]	A, B
/EC2/VirtualMachine[@select="1"]/InstanceType/Cluster/Cluster4XL	A, B
/EC2/VirtualMachine[@select="1"]/OS/Linux[text()="StdLinux"]	A, B
/EC2/VirtualMachine[@select="1"]/InstanceType/HighCPU[@cuRightBound=3]	A
/EC2/Storage[@selected="1"][text()="Std"]	A
/EC2/VirtualMachine[@select="1"]/InstanceType/HighMem[@RAMRightBound=8]	A
/EC2/VirtualMachine[@select="1"]/InstanceType/HighMem[@RAMRightBound=16]	B
/EC2/VirtualMachine[@select="1"]/InstanceType/HighCPU[@cuRightBound=16]	B
/EC2/Storage[@selected="1"][text()="Provisioned"]	B
/EC2/LoadBalancer[@select="1"]	C
/EC2/VirtualMachine[@select="1"]/OS/Windows[text()="StdWindows"]	C
/EC2/Storage[@selected="0"]	C
/EC2/VirtualMachine[@select="1"]/InstanceType/Default/L	C

feature specified in the feature diagram. On the other hand, OS (the 6th row) is a mandatory feature and it has two alternatives, i.e., Windows and Linux.

The Yfilter's XPath query generator can generate nested and non-nested queries or features. A nested representation is an XPath representation which has preconditions. For example, */AmazonEC2[Storage[@select = "1"]/ Std]/LoadBalancer[@select = "1"]* is a nested feature which means that the tenant chooses LoadBalancer while the tenant also has selected Std for Storage. But for our mapping purpose, we only need to generate non-nested features, because all features can be covered based on the semantics of a feature diagram. As a result, it can substantially reduce the number of combinations and reduce the computation time. For this particular example, if nested features are considered, the total number of automatically generated XPath representations exceeds 6000. Without considering nested XPath, the DTD can automatically generate all possible 71 XPath representations for the combined features as shown in Figs. 2 and 4. Fig. 9 shows 48 XPath representations for Amazon EC2 using Yfilter's XPath generator from the DTD as depicted in the Appendix. For space reason, the security aspect is not included in this figure.

For our approach, the feature model of the SaaS system is first developed. A feature model for each tenant is then built. The second step can be avoided if the feature model of the SaaS system is available to the tenants. In other words, each tenant can simply select and discard particular features from the available

```
<!ELEMENT AmazonEC2 (LoadBalancer, VirtualMachine, Storage, Security)>
<!ELEMENT LoadBalancer EMPTY>
<!ATTLIST LoadBalancer select (0 | 1) #REQUIRED>
    <!ELEMENT VirtualMachine(OS, InstanceType, DetailedMonitoring,
                                                    Dedicated)>
<!ATTLIST VirtualMachine select (0 | 1) #REQUIRED>
  <!ELEMENT OS (Windows | Linux) >
    <!ELEMENT Windows (SQLStd | StdWindows | SQLWeb) >
      <!ELEMENT SQLStd EMPTY>
      <!ELEMENT StdWindows EMPTY>
      <!ELEMENT SQLWeb EMPTY>
    <!ELEMENT Linux (StdLinux | RedHat | Suse) >
      <!ELEMENT StdLinux EMPTY>
      <!ELEMENT RedHat EMPTY>
      <!ELEMENT Suse EMPTY>
```

Fig. 8. DTD representations (a partial list) for Combined Feature Diagrams in Figs. 2 and 4.

```
/AmazonEC2/Virtualmachine[@select="1"]/InstanceType/Default/XL
/AmazonEC2/Virtualmachine/OS/Linux/RedHat
/AmazonEC2/Storage[@select="1"]/Std
/AmazonEC2/LoadBalancer[@select="1"]
/AmazonEC2/Virtualmachine[@select="1"]/InstanceType/Cluster/Cluster8XL
/AmazonEC2/Virtualmachine/InstanceType/Default/XL
/AmazonEC2/Virtualmachine[@select="1"]/InstanceType/HighIO
/AmazonEC2/Virtualmachine/OS/Linux/StdLinux
/AmazonEC2/LoadBalancer[@select="0"]
/AmazonEC2/Virtualmachine/InstanceType/Cluster/Cluster8XL
/AmazonEC2/Virtualmachine[@select="1"]/Dedicated[@select="1"]
/AmazonEC2/Storage[@select="1"]/Provisioned
/AmazonEC2/Virtualmachine[@select="1"]/OS/Windows/SQLStd
/AmazonEC2/Virtualmachine[@select="1"]/Dedicated[@select="0"]
/AmazonEC2/Virtualmachine[@select="1"]/InstanceType/Cluster/GPU4XL
/AmazonEC2/Virtualmachine/InstanceType/Cluster/GPU4XL
/AmazonEC2/Virtualmachine[@select="1"]/Dedicated
/AmazonEC2/Virtualmachine[@select="1"]/InstanceType/Default/S
/AmazonEC2/LoadBalancer
/AmazonEC2/Virtualmachine[@select="1"]/InstanceType/Default/M
/AmazonEC2/Virtualmachine[@select="1"]/InstanceType/Default/L
/AmazonEC2/Virtualmachine/DetailedMonitoring
/AmazonEC2/Storage/Provisioned
/AmazonEC2/Virtualmachine/OS/Windows/SQLStd
/AmazonEC2/Virtualmachine/OS/Windows/StdWindows
/AmazonEC2/Virtualmachine/OS/Linux/Suse
/AmazonEC2/Virtualmachine/Dedicated[@select="1"]
/AmazonEC2/Virtualmachine[@select="1"]/InstanceType/HighMem
/AmazonEC2/Virtualmachine[@select="1"]/InstanceType/Cluster/Cluster4XL
/AmazonEC2/Virtualmachine/InstanceType/HighMem
/AmazonEC2/Virtualmachine[@select="0"]
/AmazonEC2/Virtualmachine/Dedicated[@select="0"]
/AmazonEC2/Storage[@select="0"]
/AmazonEC2/Virtualmachine/InstanceType/Cluster/Cluster4XL
/AmazonEC2/Virtualmachine[@select="1"]/OS/Linux/RedHat
/AmazonEC2/Virtualmachine/InstanceType/Default/S
/AmazonEC2/Virtualmachine[@select="1"]/OS/Windows/SQLWeb
/AmazonEC2/Virtualmachine/InstanceType/Default/M
/AmazonEC2/Storage/Std
/AmazonEC2/Virtualmachine/InstanceType/Default/L
/AmazonEC2/Virtualmachine[@select="1"]/OS/Linux/Suse
/AmazonEC2/Virtualmachine[@select="1"]/OS/Windows/StdWindows
/AmazonEC2/Virtualmachine[@select="1"]/InstanceType/HighCPU
/AmazonEC2/Virtualmachine/InstanceType/HighCPU
/AmazonEC2/Virtualmachine[@select="1"]/DetailedMonitoring
/AmazonEC2/Virtualmachine[@select="1"]/OS/Linux/StdLinux
/AmazonEC2/Virtualmachine/InstanceType/HighIO
/AmazonEC2/Virtualmachine/OS/Windows/SQLWeb
```

Fig. 9. Yfilter generated XPath representations from DTD.

SaaS feature model. This approach can be an easy means to specify requirements in SPL. On the other hand, our approach relies on Yfilter which in turn makes use of XPath. Hence, the selected features need to be translated to XPath expressions for Yfilter to identify the matched system features.

On the other hand, our approach is highly effective because for real systems, the number of features can be large, thousands of features whose combinations could be imposed by many and often nontrivial rules [5,12]. For such systems, the proposed matching approach can effectively facilitate the management of the configuration complexity, which is one of the crucial challenges that the simple tenant selection mechanism (e.g., using GUI) cannot address properly.

7. Conclusions and future work

Multi-tenant SaaS applications are emerging primarily due to the cost reduction through resource sharing. One of the challenges for multi-tenant SaaS systems is the rapid growing co-existing configurations across multiple tenants. In addition, multiple tenants can have diverse sets of features and constraints, including layout style, general configuration, file I/O, and workflow [6]. Currently, there is still a lack of methodical support for the development of emerging multi-tenant applications in support of co-existing tenant-specific configurations. Feature modeling and automatic matching of tenant-specific feature configurations and constraints to the cloud SaaS features plays a crucial role in the effective management of multi-tenant SaaS applications.

The main contribution of the paper is a novel approach that supports the *automatic matching* of tenant-specific feature configurations, which can be used to *facilitate* SaaS deployment, feature updates, and management. We presented a framework to provide support mechanisms for constraint-based cloud services and proposed the use of Yfilter for the matching process. Yfilter has been adopted for large problem space in various data management areas and is a highly reliable and efficient solution. We have also developed a tool to automatically generate XPath feature expressions based on feature diagrams and DTD. We applied our proposed approach to two use cases. The experimental results demonstrated the correctness and effectiveness of our proposed approach both in time and space usages, as shown in Section 5 Case Studies.

A few directions warrant future research. For our case studies, only one configuration option was generated from Yfilter. If multiple configuration options are generated by Yfilter, selection of the most suitable one can be determined based on cost or other criteria, which is currently under investigation. MCDM-based approaches [1] have been proposed for service selection. One challenge is the size of configuration space and the number of criteria could be large. After the filtering process, the number of configuration options will be significantly lower, which can facilitate an efficient adoption of a MCDM-based method. Another direction is the development of a GUI tool that allows the user/designer to select the features from SaaS and then automatically generate the DTD representations. This would be useful if the number of features is not large or there are not complex tenant-specific constraints. We are also investigating the integration of existing feature modeling tools with Yfilter to provide enhanced support. A number of feature modeling tools are available. Some examples include EMF feature model [64], SPLOT [51], and tools compared by Dammagh and Troyer [65]. SPLOT supports the creation of a feature tree in Simple XML Feature Model (SXFM) format. The tool has also been extended to even generate XPath notations [12]. In this case, there is no need to encode a feature with DTD and then generate corresponding XPath notations.

Our paper focused on SaaS, but the concept of features is generic to IaaS, PaaS, and SaaS cloud services. Some cloud providers support more than one type of cloud services, e.g., Google Cloud Platform (Google) [66], Microsoft Azure (Microsoft) [67], and Amazon AWS (Amazon) [63]. Hence, the other direction is to investigate the proposed approach for other cloud services, i.e., IaaS and PaaS, and potential integration of multiple cloud services.

Acknowledgment

This project was partially sponsored by Natural Sciences and Engineering Research Council of Canada (NSERC).

```
<!ELEMENT AmazonEC2 (LoadBalancer, VirtualMachine, Storage, Security)>
<!ELEMENT LoadBalancer EMPTY>
<!ATTLIST LoadBalancer select (0 | 1) #REQUIRED>
<!ELEMENT VirtualMachine(OS, InstanceType, DetailedMonitoring, Dedicated)>
<!ATTLIST VirtualMachine select (0 | 1) #REQUIRED>
    <!ELEMENT OS (Windows | Linux) >
        <!ELEMENT Windows (SQLStd | StdWindows | SQLWeb) >
            <!ELEMENT SQLStd EMPTY>
            <!ELEMENT StdWindows EMPTY>
            <!ELEMENT SQLWeb EMPTY>
        <!ELEMENT Linux (StdLinux | RedHat | Suse) >
            <!ELEMENT StdLinux EMPTY>
            <!ELEMENT RedHat EMPTY>
            <!ELEMENT Suse EMPTY>
    <!ELEMENT InstanceType (Default | HighMem | HighIO | HighCPU | Cluster) >
        <!ELEMENT Default (S | M | L | XL) >
            <!ELEMENT S EMPTY>
            <!ELEMENT M EMPTY>
            <!ELEMENT L EMPTY>
            <!ELEMENT XL EMPTY>
        <!ELEMENT HighMem EMPTY>
        <!ELEMENT HighIO EMPTY >
        <!ELEMENT HighCPU EMPTY >
        <!ELEMENT Cluster (Cluster4XL|GPU4XL| Cluster8XL) >
            <!ELEMENT Cluster4XL EMPTY >
            <!ELEMENT GPU4XL EMPTY >
            <!ELEMENT Cluster8XL EMPTY >
    <!ELEMENT DetailedMonitoring EMPTY >
    <!ELEMENT Dedicated EMPTY >
    <!ATTLIST Dedicated select    (0 | 1)   #REQUIRED >
<!ELEMENT Storage (Provisioned | Std) >
<!ATTLIST Storage  select (0 | 1)   #REQUIRED >
     <!ELEMENT Provisioned EMPTY>
     <!ELEMENT Std EMPTY>
<!ELEMENT Security (FrontEnd, Server, InfraService, Virtualization, DataCentre) >
<!ELEMENT FrontEnd (Authentication)>
    <!ELEMENT Authentication (OAuthAuthorization | SocialSignOn | FederatedAuthentication | Password)>
        <!ELEMENT OAuthAuthorization EMPTY>
        <!ELEMENT SocialSignOn EMPTY>
        <!ELEMENT FederatedAuthentication EMPTY>
        <!ELEMENT Password (PasswordFormat+, MaximumInvalidLoginAttempts+, PasswordChangeFrequency)>
            <!ELEMENT PasswordFormat EMPTY>
            <!ELEMENT MaximumInvalidLoginAttempts EMPTY>
            <!ELEMENT PasswordChangeFrequency EMPTY>
<!ELEMENT Server (SessionManagement, DataIsolation, UserAccessAuthorization, RoleSecurityManagement)>
    <!ELEMENT SessionManagement EMPTY>
    <!ELEMENT DataIsolation (SharedSchema?, SeparateSchema?)>
        <!ELEMENT SharedSchema EMPTY>
        <!ELEMENT SeparateSchema EMPTY>
    <!ELEMENT UserAccessAuthorization (TableSecurity, RecordSecurity)>
        <!ELEMENT TableSecurity (Create | Read | Update | Delete)>
            <!ELEMENT Create EMPTY>
            <!ELEMENT Read EMPTY>
            <!ELEMENT Update EMPTY>
            <!ELEMENT Delete EMPTY>
        <!ELEMENT RecordSecurity (Read | Write | Transfer)>
            <!ELEMENT Read EMPTY>
            <!ELEMENT Write EMPTY>
            <!ELEMENT Transfer EMPTY>
    <!ELEMENT RoleSecurityManagement EMPTY>

<!ELEMENT InfraService (DataSecurity, SecureDataTransmission)>
    <!ELEMENT DataSecurity (DataLocation, DataPrivacy, DataAccessAuthorization)>
        <!ELEMENT DataLocation EMPTY>
        <!ELEMENT DataPrivacy EMPTY>
        <!ELEMENT DataAccessAuthorization EMPTY>
    <!ELEMENT SecureDataTransmission (SSL | TLS)>
        <!ELEMENT SSL EMPTY>
        <!ELEMENT TLS EMPTY>
<!ELEMENT Virtualization (PatchManagement?, SecureConfiguration?, OSHardening?)>
     <!ELEMENT PatchManagement EMPTY>
     <!ELEMENT SecureConfiguration EMPTY>
     <!ELEMENT OSHardening EMPTY>
<!ELEMENT DataCentre (PhysicalSecurity?, ElectronicAccess?) >
     <!ELEMENT PhysicalSecurity EMPTY>
     <!ELEMENT ElectronicAccess EMPTY>
```

Appendix

The complete list of DTD representations for the combined 71 features depicted in Fig. 2 (security for Saleforce.com) and Fig. 4 (Amazon EC2).

References

[1] L. Sun, H. Dong, O.L. Hussain, F.K. Hussain, E.E. Chang, Cloud service selection: state-of-the-art and future research directions, J. Netw. Comput. Appl. 45 (2014) 134–150.

[2] Salesforce.com, 2018 https://www.salesforce.com (last accessed 20.10.18).

[3] L.P. Tlzzei, M. Nery, V.C.V.B. Segura, R.F.G. Cerqueira, Using microservices and software product line engineering to support reuse of evolving multi-tenant SaaS, in: Proceedings of the 21st ACM International Systems and Software Product Line Conf, 2017, pp. 205–214.

[4] C.-P. Bezemer A. Zaidman, A, Multi-tenant SaaS applications: Maintenance dream or nightmare? in: Proceedings of IWPSE-EVOL, 2010, pp. 88–92.

[5] A. Metzger, K. Pohl, Software product line engineering and variability management: achievements and challenges, in: Proceedings of the Future on Software Engineering, 2014, pp. 70–84.

[6] S. Walraven, D. Van Landuyt, E. Truyen, K. Handekyn, W. Joosen, Efficient customization of multi-tenant software-as-a-service applications with service lines, J. Syst. Softw. 91 (2014) 48–62.

[7] P. Helland, Condos and clouds, Commun. ACM 56 (1) (2013) 50–59.

[8] D. Benavides, S. Segura, P. Trinidad, A.R.Cortes, FAMA: Tooling a framework for the automated analysis of feature models, in: Proceedings of 1st International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS), 2007, pp. 129–134.

[9] J.E. Hajlaoui, M.N. Omri, D. Benslimane, A QoS-aware approach for discovering and selecting configurable IaaS cloud services, Int. J. Comput. Syst. Sci. Eng. 32 (4) (2017) 1–16.

[10] Y. Xiang, Y. Zhou, Z. Zheng, M. Li, Configuring software product lines by combining many-objective optimization and SAT solvers, ACM Trans. Softw. Eng. Methodol. 26 (4) (2018) 1–47.

[11] L. Chen, M.A. Babar, A survey of scalability aspects of variability modeling approaches, in: Proceedings of the 13th IEEE International Software Product Lines Conference (SPLC), 2009, pp. 1–8.

[12] A. Hubaux, P. Heymans, P.-Y. Schobbens, D. Deridder, Supporting multiple perspectives in feature-based configuration, Softw. Syst. Model. 12 (3) (2013) 641–663.

[13] Y. Cao, C.-H. Lung, S. Ajila, Constraint-based multi-tenant SaaS deployment using feature modeling and XML filtering techniques, in: Proceedings of 12th International Workshop on Software Cybernetics in collaboration with the 39th Annual IEEE International Computer Software and Applications Conf, 2015, pp. 454–459.

[14] Y. Diao, M. Altinel, M. Franklin, H. Zhang, P. Fischer, Path sharing and predicate evaluation for high-performance XML filtering, ACM Trans. Database Syst. 28 (4) (2003) 467–516.

[15] XPath, XML Path Language 1.0, http://www.w3.org/TR/xpath (last accessed 02.08.18).

[16] K. Kang, S. Cohen J. Hess, W. Novak, A. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, (1990) Software Engineering Institute, Carnegie Mellon University.

[17] J. Garćıa-Galán, O.F. Rana, P. Trinidad, A. Ruiz-Cortes, Migrating to the cloud: a software product line based analysis, in: Proceedings of 3rd International Conference on Cloud Computing and Services Science, 2013, pp. 416–426.

[18] K. Pohl, G. Bockle, F. van der Lindeen, Software Product Line Engineering: Foundations, Principles and Techniques, Springer-Verlag, 2005.

[19] S. Apel, C. Kästner, An overview of feature-oriented software development, J. Object Technol. (JOT) 8 (5) (2009) 49–84.

[20] K. Czarnecki, A. Wasowski, Feature diagrams and logics: There and back again, in: Proceedings of the 11th IEEE International Software Product Line Conference (SPLC), 2007, pp. 23–34.

[21] M. Bošković, G. Mussbache, E. Bagheri, D. Amyot, D. Gasevic, M.M. Hatala, Aspect-oriented feature models, in: Proceedings of MODELS Workshops, 2011, pp. 110–124.

[22] B. Tekinerdogan, K. Öztürk, Feature-driven design of SaaS architectures, software engineering frameworks for the cloud computing paradigm, in: Software Engineering Frameworks for the Cloud Computing Paradigm, 2013, pp. 189–212.

[23] R. Mietzner, A. Metzger, F. Leyman, K. Pohl, Variability modelling to support customization and deployment on multi-tenant aware software as a service applications, in: Proceedings of ICSE Workshop on Principles of Eng. Service Oriented Systems, 2009, pp. 18–25.

[24] E. Wittern, J. Kuhlenkamp, M. Menzel, Cloud service selection based on variability modeling, in: Proceedings the 10th International Conference on Service Oriented Computing, 2012, pp. 127–141.

[25] E. Wittern, C. Zirpins, Service feature modeling: modeling and participatory ranking of service design alternatives, Softw. Syst. Model. 15 (2) (2016) 553–578.

[26] C. Quinton, L. Duchien, P. Heymans, S. Mouton, E. Charlier, Using feature modelling and automations to select among cloud solutions, in: Proceedings of 3rd International Workshop on Product Line Approaches in Software. Engineering, 2012, pp. 17–20.

[27] M. Abdel-Basset, M. Mohamed, V. Change, NMCDA: a framework for evaluating cloud computing services, Future Gener. Comput. Syst. 86 (2018) 12–29.

[28] H. Moens, F.D. Truck, Feature-based application development and management of multi-tenant applications in clouds, in: Proceedings of the 18th International Software Product Line Conference, 2014, pp. 72–81.

[29] FaMa, http://www.isa.us.es/fama/ (last accessed May 2018).

[30] Flame Framework, http://www.isa.us.es/fama/?FLAME_framework, (last accessed 02.08.18).

[31] B. Sengupta, A. Roychoudhury, Engineering multi-tenant software-as-a-service systems, in: Proceedings of ICSE Workshop on Principles of Eng. Service Oriented Systems, 2011, pp. 15–21.

[32] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, F.D. Truck, Cost-effective feature placement of customizable multi-tenant applications in the cloud, J. Netw. Syst. Manag. 22 (4) (2014) 517–558.

[33] J. J. Garćıa-Galán, P. Trinidad, Modelling and analysing highly-configurable services, in: Proceedings of the 21st Int'l Systems and Software Product Line Conference – Volume A, 2017, pp. 114–122.

[34] W. Alwis, C. Gamage, A multi-tenancy aware architectural framework for SaaS application development, J. Inst. Eng. 46 (3) (2013) 21–31.

[35] M. Dutta, P. Gupta, Customization issues in cloud based multi-tenant SaaS applications, Int. J. Eng. Comput. Sci. 3 (4) (2014) 5447–5452.

[36] J. Espadas, A. Molina, G. Jimenez, M. Molina, P. Ramirez, D. Concha, A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures, Future Gener. Comput. Syst. 29 (2011) 273–286.

[37] S. Walraven, E. Truyen, W. Joosen, A Middleware Layer for Flexible and Cost-efficient Multi-Tenant Applications, in: Proceedings of the 12th International Conference on Middleware, 2011, pp. 370–389.

[38] W.-T. Tasi, P. Zhong, Y. Chen, Tennat-centric sub-tenancy architecture in software-as-a-service, CAAI Trans. Intell. Technol. 1 (2) (2016) 150–161.

[39] XML, Extensible Markup Language, http://www.w3.org/XML (last accessed 02.08.18).

[40] JSON, Introducing JSON, http://json.org/ (last accessed 02.08.18).

[41] E.R. Harold, W.S. Means, XML in a Nutshell, O'Reilly, 2004.

[42] K. Candan, W.-P. Hsiung, S. Chen, D. Agrawal, AFilter: Adaptable XML filtering with prefix-caching and suffix-clustering, in: Proceedings of International Conference on VLDB, 2006, pp. 559–570.

[43] L. Dai, C.-H. Lung, S. Majumdar, BFilter: Efficient XML message filtering and matching in publish/subscribe systems, J. Softw. 11 (4) (2016) 376–402.

[44] S. Chen, H.G. Li, J. Tatemura, W.-P. Hsiung, Scalable filtering of multiple generalized-tree-pattern queries over XML streams, IEEE Trans. Knowl. Data Eng. 20 (12) (2008) 1627–1640.

[45] W. Sun, Y. Qin, P. Yu, Z. Zhang, Z. He, HFilter: Hybrid finite automaton based stream filtering for deep and recursive XML data, database and expert systems applications, in: Lecture Notes on Computer Science, Vol. 5181, 2008, pp. 566–580.

[46] P. Saxena, R. Kamal, System architecture and effect of depth of query on XML document filtering using PFilter, in: Proceedings of International Conference on Contemporary Computing, 2013, pp. 192–195.

[47] J. Kwon, P. Rao, B. Moon, S. Lee, Fast XML document filtering by sequencing twig patterns, ACM Trans. Internet Technol. 9 (4) (2009) 1–51.

[48] D. Lee, H. Shin, J. Kwon, W. Yang, S. Lee, Sfilter: Schema based filtering system for XML streams, in: Proceedings of the International Conference on Multimedia and Ubiquitous Engineering, 2007, pp. 266–271.

[49] R. Martins, J. Pereira, Wfilter: Efficient XLM _Filtering for Large Scale Publish/Subscribe Systems, in: Proceedings of Symp. on INForum, 2011, pp. 1–14.

[50] V. Cechticky, A. Pasetti, O. Rohlik, W. Schaufelberger, XML-based feature modelling software reuse: Methods techniques tools, in: Lecture Notes in Computer Science, Vol. 3107, 2004, pp. 101–114.

[51] SPLOT, 2016 Software Product Lines Online Tools, http://www.splot-research.org/ (last accessed 20.10.18).

[52] J. Garćıa-Galán, L. Pasquale, P. Trinidad, A. Ruiz-Cortes, User-centric adaptation analysis of multi-tenant services, ACM Trans. Autonom. Adapt. Syst. 10 (4) (2016) 24, pp. Jan. 2016.

[53] T. Zheng, X. Zheng, Y. Zhang, Y. Deng, E. Dong, R. Zhang, X. Liu, SmartVM: a SLA-aware microservice deployment framework, World Wide Web (2018) 1–19.

[54] J. White, D. Schmidt, D. Benavides, P. Trinidad, Automated diagnosis of product-line configuration errors in feature models, in: Proceedings of 12th International. Software Product Line Conference, 2008, pp. 225–234.
[55] W.-C. Hsu, I.-E. Liao, CIS-X: A compacted index scheme for efficient query evaluation of XML documents, Inform. Sci. 241 (2013) 195–211.
[56] F. Li, H. Wang, L. Hao, J. Li, H.H. Gao, Approximate joins for XML at label level, Inform. Sci. 282 (2014) 237–249.
[57] J. Liu, Z.M. Ma, L. Yan, Efficient labeling scheme for dynamic XML trees, Inform. Sci. 221 (2013) 338–354.
[58] X. Liu, L. Chen, C. Wan, D. Liu, X. Xiong, Exploiting structures in keyword queries for XML search, Inform. Sci. 240 (2014) 56–71.
[59] J. Lu, T.W. Ling, Z. Bao, C. Wang, Extended XML tree pattern matching: theories and algorithms, IEEE Trans. Knowl. Data Eng. 23 (3) (2011) 1–15.
[60] S. Madria, Y. Chen, K. Passib, S. Bhowmick, Efficient processing of XPath queries using indexes, Inf. Syst. 32 (2017) 131–159.
[61] A. Termehchy, M. Winslett, Using structural information in XML keyword search effectively, ACM Trans. Database Syst. 36 (1) (2011) 1–49.
[62] S. Urli, M. Blay-Fornarino, P. Collet, Handling complex configurations in software product lines: a tooled approach, in: Proceedings of the IEEE 18th International Software Product Line Conference (SPLC), 2014, pp. 112–121.
[63] AWS, 2016, Amazon Web Services, https://aws.amazon.com/ (last accessed 16.11.16).
[64] EMF, Feature Model, Eclipse Modeling Framework Technology Project (EMFT) http://www.eclipse.org/proposals/feature-model/ (last accessed 02.08.18).
[65] M.E. Dammagh, L.D. Troyer, Feature modeling tools: evaluation and lessons learned, in: Advances in Conceptual Modeling, Recent Developments and New Directions, in: Lecture Notes in Computer Science, vol. 6999, Springer, 2011, pp. 120–129.
[66] Google Cloud Platform, https://cloud.google.com/ (last accessed 13.08.18).
[67] Microsoft Azure, (2018) https://azure.microsoft.com/ (last accessed 02.08.18).

Yang Cao received the B.S. and M.S. degrees in Computer Science and Engineering from Northeastern University, Shenyang, China and the Ph.D. degree in Computer Science from Carleton University, Ottawa, Canada. She worked as Postdoc in University of Illinois Urbana-Champaign from 2015 to 2017. She is working in Microsoft Beijing as a Software Engineer in the big data and AI platform team.

Chung-Horng Lung received the B.S. degree in Computer Science and Engineering from Chung-Yuan Christian University, Taiwan and the M.S. and Ph.D. degrees in Computer Science and Engineering from Arizona State University. He was with Nortel Networks from 1995 to 2001. In September 2001, he joined the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he is now a Professor. His research interests include: Software Engineering, Cloud Computing, and Communication Networks.

Samuel A. Ajila received B.Sc. (Hons.) in Computer Science from University of Ibadan, Nigeria, M.Sc. in Computer Science from University of Ife, Nigeria, DEA and Ph.D. in Computer Science and Engineering from LORIA, Université de Lorraine, Nancy, France. He was with Nelson Mandela University, Port Elizabeth, South Africa as a Senior Lecturer and HOD in Computer Science and Information Systems from 1998 to 2001. In September 2001, he joined the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada where he is currently an Associate Professor. His research interests include Software Engineering, Cloud Computing, Machine Learning, and Big Data Analytics.

Xiaolin Li received the B.S degree in Electronic Information Engineering from Sanxi University, Shanxi, China and the M.S. in Computer Science from Shanghai Jiaotong University, Shanghai, China and the second M.A.Sc. in System and Computer Engineering from Carleton University, Ottawa, Canada. She is working as a Software Engineer at Cisco, Ottawa, Canada.