2015 IEEE 39th Annual International Computers, Software & Applications Conference

Constraint-based Multi-tenant SaaS Deployment Using Feature Modeling and XML Filtering Techniques

Yang Cao, Chung-Horng Lung, Samuel Ajila
Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada
Email: ycao5@scs.carleton.ca, {chlung, ajila}@sce.carleton.ca

***Abstract* — Software-as-a-service (SaaS) is becoming more important as a software delivery and service model. However, multi-tenancy, which promises to provide a high degree of resource sharing among a large number of tenants (customers or organizations), can significantly complicate SaaS development, deployment, and management due to potential explosion of co-existing tenant-specific variations. Manually configuring those tenant-specific variations for multi-tenant SaaS systems cannot satisfy scalability and flexibility. In this paper, we propose a novel approach in support of multi-tenant SaaS systems using feature modeling and XML filtering techniques. Feature modeling is used to capture functional, non-functional requirements and constraints. The features of a cloud system and tenant-specific requirements are encoded with XPath feature representations and XML document, respectively. We adopt Yfilter, an established XML filtering technique, and tailor it to match cloud configurations that satisfy tenant-specific requirements and constraints. The experimental results demonstrate that our approach can automatically and correctly identify cloud system configurations that match tenant-specific requirements. In addition, the execution time in our approach is only a small fraction compared to the existing approaches (e.g., FaMa) and the configuration space is also smaller.**

***Keywords*—Cloud Computing, Software-as-a-Service, Feature Modelling, XML Filtering, Yfilter**

I. INTRODUCTION

Cloud computing is a term used for on-demand computing services with shared-access of configurable computing services. There are three cloud services: infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS). In addition, there are three cloud deployment models: public, private, and hybrid. Cloud computing provides elastic services with lower cost in a pay-per-use model. Cloud services can be used by individual users or multiple tenants, where a tenant is an entity or an organization with a group of users sharing the same software system.

Multi-tenancy enables the SaaS providers to achieve a large scale of resource sharing. Multi-tenancy is different from multi-user or multi-instance [10]. In multi-tenancy, each tenant may have a high degree of configurability of the software system. Further, the number of tenants may be too large to be effective using the multi-instance approach. Some key features of multi-tenancy include hardware resource sharing, large configuration space, and a limited number of instances (or even a single instance) of a single application and database. From the multi-tenancy and service-level agreements (SLAs) perspectives, the possibility of having different workflows for the same application is very high for different tenants [10], and even different versions may be needed for the same tenant [2]. As a result, a number of challenges for multi-tenancy arise. Scalability, operational costs, flexibility, and security, among others, are key considerations for multi-tenant SaaS systems development, deployment, and management [2].

Deployment of software systems and/or components on clouds can be performed using different approaches: (1) coarse-grained partitioning over different clouds according to the requirements; (2) application of software product line (SPL) technologies to support different variants and configurations; and (3) consideration for data placement, cost, and other non-functional constraints [1]. The challenges associated with development, deployment, and management for cloud computing, particularly multi-tenancy, are primarily caused by resource sharing, a large number of possible configurations [2, 10], and various constraints, e.g., cloud type or location. For SaaS development, typically, requirements gathering and system design are conducted in an a priori, rather than specific to a tenant. In other words, a pre-packaged and configurable system [3] is provided to tenants, rather than having to conduct traditional requirements gathering for each tenant. In addition, there are the difficulties of managing the tradeoffs over various non-functional dimensions such as performance, security, privacy, availability, and cost.

The main contributions of this paper are summarized as follows. First, we define a reference framework for constraint-based multi-tenant approach for SaaS development, deployment, and management. Second, using the reference model, we adopt and tailor the XML-based techniques for: (1) encoding the feature models with XPath feature representations and representing tenant-specific configurations in an XML document; and (2) applying the well-known Yfilter [6] filtering and matching technique in XML data processing to the target problem. Thus, one can automatically select appropriate cloud configuration or specific service products from a large set of possible options. Our approach can do two things: generates cloud configurations that meet a tenant's requirements and constraints, and also significantly reduces the number of configuration space and the computation time.

The remainder of this paper is structured as follows. Section II describes the related works. Section III presents an overview of the reference framework used in this research. Section IV demonstrates the automatic transformation and

0730-3157/15 $31.00 © 2015 IEEE
DOI 10.1109/COMPSAC.2015.255

IEEE computer society

Authorized licensed use limited to: Fachhochschule Harz. Downloaded on April 04,2022 at 13:37:24 UTC from IEEE Xplore. Restrictions apply.

matching technique for multi-tenant feature configurations. Section V illustrates two case studies based on the proposed approach. Finally, section VI concludes the paper and discusses some future research directions.

II. RELATE WORKS

This section first describes related research in multi-tenant software systems and some key ideas. The second subsection highlights the related techniques in XML that are used in our research and the third section is a summary.

A. Multi-tenant SaaS

While there is a great deal of research on multi-tenant applications and deployment, the challenge due to a large number of tenant configuration space for multi-tenancy has not been adequately explored. Most of the existing works on multi-tenant SaaS have focused on shared infrastructure (i.e., virtualization techniques), data architecture, security management, and middleware extensions [7].

Bezemer and Zaidman [10] described the multi-tenant approach and compared it with multi-user and multi-instance approaches. Further, the authors identified the challenges in the multi-tenant approach. Walraven et al, [2] discussed the multi-tenant systems and challenges in details. They identified a lack of methodical support for the development of multi-tenant applications and presented an approach in support of co-existing tenant-specific configurations.

Feature modeling [14] is one of the key techniques in the service line engineering [2] method for multi-tenant SaaS cloud applications. In particular, the work in [16] extended variability modeling for a tenant to customize the SaaS application. Feature model was used in [17] to select the best cloud service (PaaS or IaaS) provider for a given application. The authors in [27] presented a theoretical analysis on three approaches for the management of multi-tenant SaaS software. Based on their results, the feature-based approach generated fewer runtime instance types.

While most papers focused on methodology or theoretical evaluation, [17] and [4, 5, 18] presented a tool support in cloud computing. The tool merges feature models of all available PaaS solutions into an aggregated feature model [17]; however, the authors did not show how to identify a cloud provider based on feature models automatically. The works in [18] and [28] analyzed a feature model using FaMa. This tool translates a feature model to a representation (SAT, CSP, or BDD) and used a corresponding solver to perform analysis on the feature model. Although FaMa is a framework for analyzing feature models, it generates a large number of configuration options that needs further refinement and it has not been demonstrated to analyze medium to large size feature models in a reasonable time.

The authors in [8] developed a tenant-based resource allocation model for multi-tenant SaaS applications. In [9], a middleware layer was implemented to handle customization and multi-tenancy. The work in [10] presented an overview on the challenges in multi-tenant SaaS systems. In [7] and [10], an 3-tier architecture (authentication, configuration, and database) was adopted from the traditional single-tenant web application to support multi-tenancy. The work of Nitu [11] stored configuration data with XML and injected them into applications at runtime. Authors in [12] used XML to store tenant's customization data in a database. Alwis and Gamage [13] presented a framework in which the model-driven approach was used to develop SaaS applications.

B. XML, XPath, and XML filtering techniques

XML [19] represents structured information in a machine-readable format. XFeature [20] is an XML-based feature modeling tool that supports the modeling of product families and instantiates applications from the model. The authors in [20] used XML to represent a feature diagram. But this tool does not support the automated analysis of feature diagrams. XPath [21] is a language for retrieving, selecting, and filtering information from an XML document. XML filtering is an established technique that has been thoroughly investigated to match an XML document *d* against a number of XPath feature (query) representation *F* to identify whether *d* has the isomorphic tree structure and values defined in *F*. Advanced XML filtering algorithms have been studied extensively (Yfilter [6], Afitler [22], Bfilter [23] and Gfilter [24]).

C. Discussion

Multi-tenancy has the potential to significantly reduce cost, as multiple users make use of the same application instance and share resources. However, in spite of the importance of multi-tenancy, there is a lack of support for different requirements made by different tenants. As a result, either a one-size-fits-all approach [2] or an application-based approach [27] is common in practice. The one-size-fits-all approach does not satisfy tenant-specific requirements and the application-based approach generates a large number of application instances. For multi-tenancy, service-level agreements (SLAs), the workflow may be different for different tenants and versioning may be needed for the same tenant. As a result, the number of variations or configurations can grow rapidly.

In comparison to the approaches using the formal method [4], the configuration space in our method is much smaller because common paths for features in a feature diagram are shared using Yfilter, instead of enumerating all possible product options. The processing time is also very small, which is suitable for potentially large numbers of features and tenants.

III. CONSTRAINT-BASED CLOD DEPLOYMENT FRAMEWORK

This section briefly presents an overview of our conceptual constraint-based multi-tenant approach for cloud deployment. The framework is a modification of that presented in [2]. As depicted in Fig. 1, our framework consists of four processes (shown in round rectangles). Due to the page limit, only a few components are highlighted in the following.

During the process of *Tenant Requirements Analysis and Configuration Management* (bottom left in Fig. 1), the tenant requirements (both functional and non-functional) and constraints are identified and analyzed. For SaaS,

Authorized licensed use limited to: Fachhochschule Harz. Downloaded on April 04,2022 at 13:37:24 UTC from IEEE Xplore. Restrictions apply.

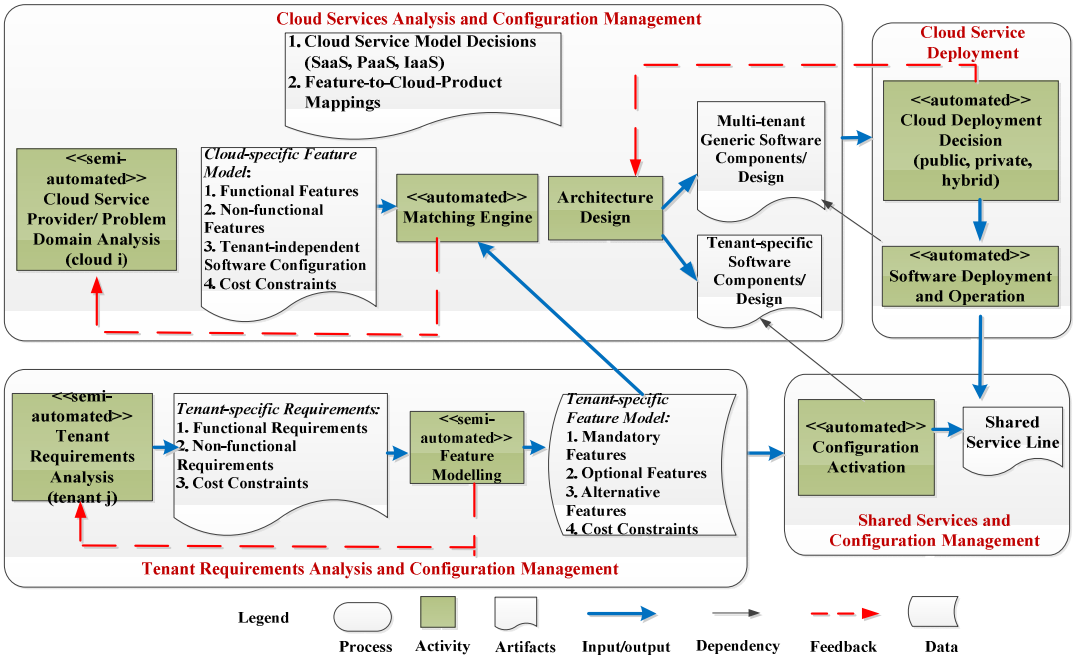

Fig. 1. Constraints-based Cloud Deployment Framework

requirements and constraints typically are gathered from tenants through the SaaS provider's configuration tool. Then, these requirements and constraints are processed through the activity *Feature Modeling*. The tenant-specific features are later encoded in XML format.

The *Cloud Services Analysis and Configuration Management* (top left in Fig. 1) models cloud services/ products using feature diagrams. The feature diagram for a cloud system includes functional and non-functional features, tenant-independent software configurations, and cost constraints. The cloud-specific feature diagrams and the tenant-specific XML documents (obtained from the previous process) are the inputs to the activity *Matching Engine* which uses Yfilter to select the appropriate cloud service products that match the tenant-specific requirements.

IV. Automatic XML-Based Matching Engine

This section presents the matching technique which can automate the selection of tenant-specific configuration. The proposed approach consists of five steps:

1. Model the cloud system features with feature diagrams
2. Encode each feature diagram of the cloud system with XPath feature representations
3. Model the tenant-specific requirements with feature diagrams
4. Encode the tenant-specific feature diagrams with XML format
5. Apply the Yfilter matching technique to identify the matched system features based on the tenant-specific requirements and constraints.

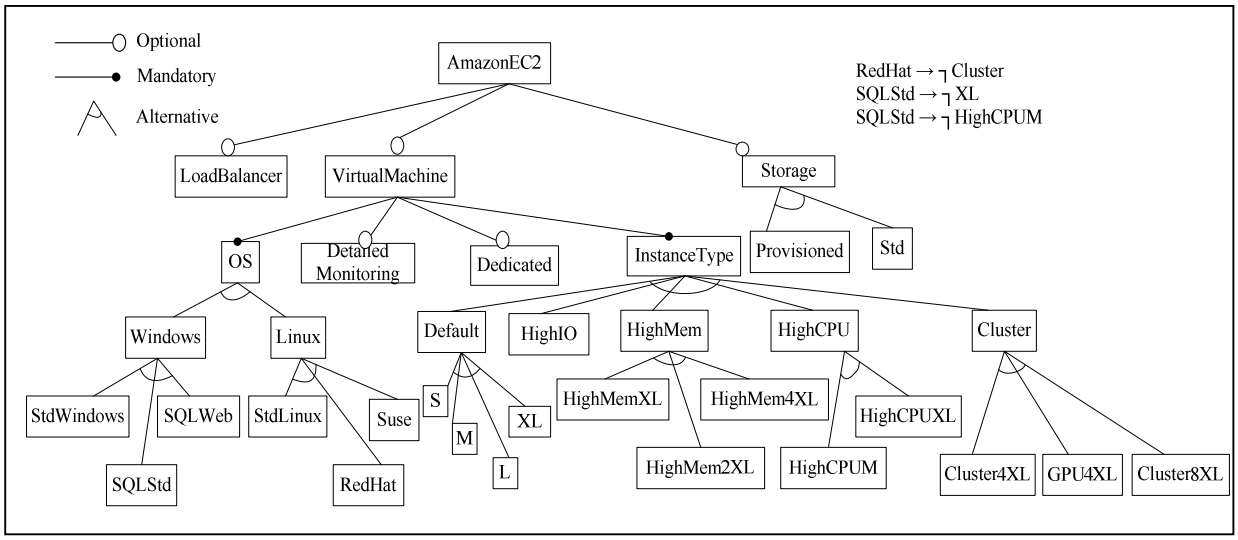

Fig. 2. Feature Diagram for Amazon EC2 [4]

Authorized licensed use limited to: Fachhochschule Harz. Downloaded on April 04,2022 at 13:37:24 UTC from IEEE Xplore. Restrictions apply.

A. *Perform feature modeling*

Both step 1 and step 3 are similar to the traditional feature modeling process. We demonstrate a feature diagram based on Amazon EC2 as an example, as shown in Fig. 2, which is used to present how the feature diagram can be represented as a set of XPath feature representations, as depicted in TABLE I.

TABLE I. XPath Feature Representation for Fig. 2

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

B. *Apply XML-based matching technique*

This section presents our XML-based matching technique to automatically select a set of cloud configurations that satisfy the tenant-specific requirements and constraints.

In our approach, XPath feature representations are used to encode the feature diagram(s) and the proven Yfilter technique is used for XML document matching to identify tenant-specific configuration for cloud deployment.

In the XML-based matching approach, 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) (Fig. 3). An XML document which contains the configuration for each tenant is matched against the NFA using the Yfilter technique. The generated results are a set of cloud features that satisfy the tenant's configuration. In step 5, the Yfitler technique is applied to identify the matched system features based on the tenant-specific requirements and constraints. [6].

Fig. 3 depicts an example of three features XPath representations and the NFA representation used in Yfilter. The NFA tree covers three features F0, F1, and F2. Each circle in the figure denotes a state. Two concentric circles denote an accepting state; such states are associated with the IDs of the features they represent. A directed edge with a solid arrow represents a transition. The symbol on an edge represents the input that triggers the transition. The special symbol "$*$" 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.

SAX [26] is used to drive state transitions of the automata to process the filtering and matching task. The aggregated XPath feature representations in Fig. 3 can be

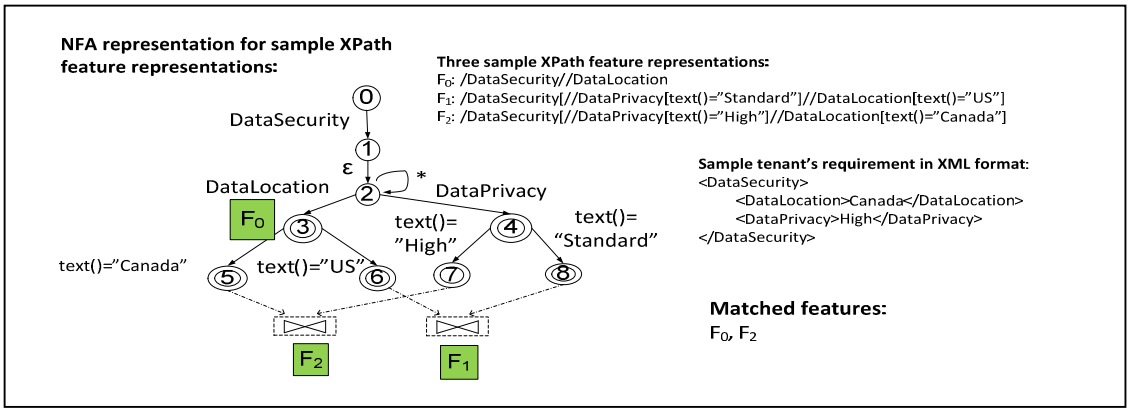

Fig. 3. Three Sample XPath Feature Representations and a Sample Tenant's Requirements in XML Format

Authorized licensed use limited to: Fachhochschule Harz. Downloaded on April 04,2022 at 13:37:24 UTC from IEEE Xplore. Restrictions apply.

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.

TABLE II. List of Selected Features for Tenants

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: optional Virtual Machine: mandatory; Memory: 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

C. *Validation of the proposed approach*

Our proposed solution uses feature modeling and the XML matching technique. The SaaS features are converted to XPath expressions and tenant-specific requirements and constraints are translated into XML documents. It has been proven that Yfilter can correctly generate the matching results based on two inputs, XPath and XML [6], and it has been adopted for various applications in information processing. Hence, our approach to SaaS application (using XML and Yfilter) also can generate correct matching results.

A key benefit of using an NFA-based approach is the tremendous reduction in the space size. The space complexity in our approach is $O(n)$, where n is the number of nodes in a cloud-side feature diagram. For time complexity, the number of states in NFA can increase fast when a recursive element exists in document, though recursion has not yet been used in our application. However, the performance of Yfilter in general is still highly efficient compared to the approach using logical solver. The next section presents some experimental results. For multi-tenant SaaS systems, the number of tenants alone can be thousands or higher, hence the total number of features that need to be managed can be much higher. As cloud software keeps evolving, system features tend to grow exponentially. Since tenant requirements usually change over time, 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[2].

V. CASE STUDIES

We have conducted experiments using a subset of features for Amazon EC2 service and Salesforce.com CRM system. This section presents one of the case studies as an example.

In [4], the authors modeled Amazon EC2, EBS, S3 and RDS with feature diagrams as shown in Fig. 2. The feature diagram was analyzed with FaMa [4]. This small feature diagram yields 1,758 different valid product options as generated by FaMa. Moreover, the execution time for identifying only one valid partial configuration from this feature diagram is 135 ms [4]. The execution time for the optimization operation is even more than one hour. As more features are considered, the matching task becomes more complex. The high execution time using the approach in [4] is caused by enumerating all possible configurations offered by a feature diagram one by one.

Using our approach, the encoded XPath feature expressions for Fig. 2 is listed in TABLE I, which contains 30 features in total. There are three tenants used in the experiment.

The requirements of tenant A are captured in XML format, as illustrated in Fig. 4 as an example. TABLE II summaries the features specified for tenants A, B, and C. Yfilter is then applied to find the matched features provided by the SaaS. As expected, the resulting features identified by Yfilter for tenant A, as depicted in the bottom of Figure 4, are consistent with the features listed in TABLE II. Results for tenants B and C are also correct.

In comparison the configuration space in our approach is much smaller than the approach presented in [4], because the common paths for features in a feature diagram are shared using Yfilter, instead of enumerating all possible product options. The processing time is also very small, which is suitable for potentially large numbers of features and tenants. In our experiment, Yfilter only takes around 5 ms for the matching operation. The measured processing time is given in TABLE III. The processing time is much lower than the time taken (~ 135 msec) using the method presented in [4].

Yfilter can be much more scalable when the number of features is large. For example, the processing time of Yfilter is only around 30 msec when the number of queries is slightly larger than 50,000 [6]. In comparison, in [5], the authors showed a case study based on the diagnosis of over 5,000 features which is considered a very large number for the traditional SPLs. The processing time in this case study is ~50 seconds which is considerably higher than Yfilter. (Note that diagnosis of features may require different computational cost, but the result reveals the range of time it needs.)

Table III. Processing Time for the EC2 Example

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

VI. CONCLUSIONS AND FUTURE WORK

Multi-tenant SaaS applications are emerging due to the cost reduction through resource sharing. One of the central challenges in this area is the rapid growing of both co-existing configurations across multiple tenants and the number of tenants. The pre-packaged and highly configurable system is a key characteristic for multi-tenancy SaaS [3]. Automatic transformation or mapping of tenant-specific feature configurations and constraints to the cloud SaaS features hence plays a crucial role in the management of multi-tenant SaaS applications as the rapid increase of both the number of features and tenants. The main contribution of this paper is to present an

Authorized licensed use limited to: Fachhochschule Harz. Downloaded on April 04,2022 at 13:37:24 UTC from IEEE Xplore. Restrictions apply.

approach that supports the *automatic and efficient mapping* of tenant-specific feature configurations to SaaS provider's configuration features. Yfilter has been adopted and adapted for our approach for its robustness and efficiency. The experimental results demonstrated the correctness and effectiveness of our proposed approach both in time and space complexity.

Automatic generation of XPath feature expressions from Document Type Definitions (DTD) or XML schemas is under investigation because there are many XML generation packages for different programming languages and XML validation tools which can validate XML message against XML DTD or schema.

```
*** 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. 4. Requirements and Matching Results for Tenant A

REFERENCES

[1] A. Alkhalid, C.-H. Lung, and S. Ajila, "Towards Efficient Software Deployment in the Cloud Using Requirements Decomposition", *Proc. of the Workshop on Requirements Engineering*, pp.100–105, 2013.

[2] S. Walraven, et al., "Efficient Customization of Multi-tenant Software-as-a-Service Applications with Service Lines", *Journal of Syst. and Software*, vol. 91, pp.48-62, 2014.

[3] P. Helland, "Condos and Clouds", Commun. ACM, vol. 56, no. 1, pp.50–59, 2013.

[4] J. Garćıa-Galán, et al., "Migrating to the Cloud: a Software Product Line Based Analysis", *Proc of 3rd Int'l Conf. on Cloud Computing and Services Science*, pp.416–426, 2013.

[5] J. White, et al., "Automated Diagnosis of Product-line Configuration Errors in Feature Models", *Proc. of 12th Intl.' Software Product Line Conf*., pp.225–234, 2008.

[6] Y. Diao, M. Altinel, et al., "Path Sharing and Predicate Evaluation for High-Performance XML Filtering", *ACM Trans. Database Syst*., 28 (4), pp.467–516, 2003.

[7] B. Sengupta, A. Roychoudhury,"Engineering Multi-Tenant Software-as-a-Service Systems", *Proc. of ICSE PESOS*, 2011.

[8] J. Espadas, et al., "A Tenant-Based Resource Allocation Model for Scaling Software-as-a-Service Applications over Cloud Computing Infrastructures", *Future Generation Computer Systems,* vol. 29, pp.273–286, 2011.

[9] S. Walraven, et al., "A Middleware Layer for Flexible and Cost-efficient Multi-Tenant Applications", *Proc. of the 12th Int'l Conf. on Middleware*, pp. 370–389, 2011.

[10] C.-P. Bezemer, A. Zaidman, "Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?", *Proc. of IWPSE-EVOL*, pp.88–92, 2010.

[11] Nitu, "Configurability in SaaS (Software as a Service) Applications", *Proc. of India Sw. Eng. Conf*., pp.19–26, 2009.

[12] M. Dutta, P. Gupta, "Customization Issues in Cloud Based Multi Tenant SaaS Applications", *Int'l Journal of Engineering and Computer Science*, 3 (4), pp. 5447–5452, 2014.

[13] W.N.T. Alwis and C.D. Gamage, "A Multi-tenancy Aware Architectural Framework for SaaS Application Development", J. of the Institution of Engineers, 46(3), pp.21–31, 2013.

[14] K. Pohl, et al., Software Product Line Engineering: Foundations, Principles and Techniques, Springer-Verlag, 2005.

[15] B. Tekinerdogan and K. Öztürk, "Feature-Driven Design of SaaS Architectures, Software Engineering Frameworks for the Cloud Computing Paradigm", in *Software Eng. Frameworks for the Cloud Computing Paradigm*, pp.189–212, 2013.

[16] R. Mietzner, et al.,"Variability Modelling to Support Customization and Deployment on Multi Tenant Aware Software as a Service Applications", *Proc. of ICSE PESOS*, 2009.

[17] C. Quinton, et al., "Using Feature Modelling and Automations to Select among Cloud Solutions", Proc. of 3rd Int'l Workshop on Product Line Approaches in Sw. Eng., pp.17–20, 2012.

[18] FaMa Tool, http://www.isa.us.es/fama/, last accessed in 2014.

[19] Extensible Markup Language (XML), http://www.w3.org/XML/.

[20] V. Cechticky, et al., "XML-Based Feature Modelling", *Software Reuse: Methods, Techniques, and Tools*, Lecture Notes in Computer Science Volume 3107, 2004, pp.101–114.

[21] XML Path Language (XPath) 1.0, http://www.w3.org/TR/xpath/.

[22] K.S. Candan, et al., "AFilter: Adaptable XML Filtering with Prefix-Caching and Suffix-Clustering", *Proc. of Int'l Conf on VLDB*, pp.559–570, 2006.

[23] L. Dai, C.-H. Lung and S. Majumdar, "BFilter-A XML Message Filtering and Matching Approach in Publish/Subscribe Systems", *Proc. of GLOBECOM 2010*, pp.1–6, 2010.

[24] S. Chen, et al., "Scalable Filtering of Multiple Generalized-Tree-Pattern Queries over XML Streams", *IEEE Trans on Knowledge and Data Eng,* 20 (12), Dec 2008, pp.1627–1640.

[25] M. Bošković, et al., "Aspect-Oriented Feature Models", Proc. Of MODELS 2010 Workshops, pp.110–124, 2011.

[26] SAX: Simple API for XML, http://www.saxproject.org/.

[27] H. Moens and F. De Truck, "Feature-Based Application Development and Management of Multi-Tenant Applications in Clouds", *Proc. of the 18th Int'l Software Product Line Conf*., pp. 72–81, Sept. 2014.

[28] D. Benavides, S. Segura, P. Trinidad and A.R. Cortes, "FAMA: Tooling a Framework for the Automated Analysis of Feature Models", *Proc. of 1st Int'l Workshop on Variability Modeling of Software-Intensive Systems*, pp.129–134, 2007.

Authorized licensed use limited to: Fachhochschule Harz. Downloaded on April 04,2022 at 13:37:24 UTC from IEEE Xplore. Restrictions apply.